**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

# Дипломна робота

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою «Інформаційні технології моніторингу довкілля»**

**спеціальності 122 «Комп'ютерні науки та інформаційні технології»**

**на тему: «Автоматизоване тестування кардіологічних web – додатків в хмарному середовищі»**

Виконав (-ла):
студентка IV курсу, групи ТМ-62
Тулук Анна Сергіївна _____

Керівник:
Старший викладач
Бандурка Олена Іванівна _____

Консультант:
_____

Рецензент:
доцент каф. ТЕУТ та АЕС
Сірий Олександр Анатолійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

**Національний технічний університет України**
**“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Інформаційні технології моніторингу довкілля

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

# ЗАВДАННЯ
**на дипломну роботу студенту**
Тулук Анні Сергіївні
(прізвище, ім’я, по батькові)

1. Тема роботи «Автоматизоване тестування кардіологічних web – додатків в хмарному середовищі»

керівник роботи Бандурка Олена Іванівна, старший викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. **№ 1168-с**

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи мова програмування JavaScript, середовище Visual Studio Code, тестовий фреймворк Nightmare
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати проблему створення кардіологічних продуктів без підтримки автоматизованого тестування, розглянути серцево-судинні захворювання, їх відображення на електрокардіограмі та параметри варіабельності серцевого ритму, розробити автоматизованне тестування кардіологічного web-додатку
5.Перелік ілюстративного матеріалу зображення тестованого кардіологічного web-додатку, зображення кардіограми та QRS комплексу, архітектура

системи автоматизованого тестування, скріншоти тест-репорту, який був автоматично створений на основі результатів тестування, графік зростання смертності людей з серцево-судинними захворюваннями від COVID-19____
_____

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |

7. Дата видачі завдання ”___”_____201___ р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітки |
|---|---|---|---|
| 1. | Затвердження теми роботи | 25.10.19 | |
| 2. | Вивчення та аналіз задачі | 20.11.19-09.12.19 | |
| 3. | Розробка архітектури та загальної структури системи | 10.12.19-20.01.20 | |
| 4. | Розробка структур окремих підсистем | 21.01.20-30.03.20 | |
| 5. | Програмна реалізація системи | 31.03.20-24.04.20 | |
| 6. | Оформлення пояснювальної записки | 27.04.20-29.05.20 | |
| 7. | Захист програмного продукту | 08.06.20 | |
| 8. | Передзахист | 08.06.20 | |
| 9. | Захист | | |

Студент _____ _Анна ТУЛУК_____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Олена БАНДУРКА_____
(підпис) (прізвище та ініціали,)

## АНОТАЦІЯ

Метою дипломної роботи є реалізація автоматизованого тестування кардіологічних web – додатків в хмарному середовищі. Роботу виконано на 80 аркушах, вона містить 4 додатки та перелік посилань на використані джерела з 10 найменувань, 33 рисунки та 15 таблиць. Реалізоване автоматизоване тестування повинно бути кросбраузерним та створювати тест-репорт для підведення підсумків тестових заходів та результатів. У ході реалізації автоматизованого тестування мають бути задокументовані тестові артефакти, такі як тест-план та тест-кейси.

Ключові слова: автоматизоване тестування, кардіологія, серцево-судинні захворювання, web-додаток, тестові артефакти.

## ABSTRACT

The purpose of the thesis is the implementation of automated testing of cardiac web - applications in a cloud environment. The work is done on 80 sheets, it contains 4 appendixes and a list of references to the used sources composed of 9 items, 33 figures and 15 tables. Implemented automated testing should be cross-browser and create a test report to summarize the test activities and results. Test artifacts, such as test plans and test cases, should be documented during implementation of automated testing.

Key words: automated testing, cardiology, cardiovascular diseases, web-application, test artifacts.

# ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ……..…..6
ВСТУП…….7
1 ПОСТАНОВКА ЗАДАЧІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ КАРДІОЛОГІЧНИХ WEB-ДОДАТКІВ В ХМАРНОМУ СЕРЕДОВИЩІ..... 9
2 АНАЛІЗ ПРОБЛЕМИ СТВОРЕННЯ КАРДІОЛОГІНИХ ПРОДУКТІВ БЕЗ ПІДТРИМКИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ..... 11
3 ЗАСОБИ РОЗРОБКИ..... 16
3.1 Середовище розробки Visual Studio Code ..... 16
3.2 Веб-інтерфейс..... 17
3.3 Бібліотека Nightmare..... 18
3.4 Chrome DevTools..... 19
3.5 Хмарна PaaS-платформа Heroku..... 28
4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ..... 29
4.1 Тест-план..... 29
4.2 Тест-кейси..... 38
4.3 Структура програмного забезпечення..... 56
5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ..... 60
5.1 Системні вимоги..... 60
5.2 Робота користувача з програмним продуктом..... 60
ВИСНОВКИ..... 69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 70
ДОДАТОК 1..... 71
ДОДАТОК 2..... 73
ДОДАТОК 3..... 81
ДОДАТОК 4..... 94

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) — набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

ССЗ — серцево судинні захворювання.

ЕКГ — електрокардіограма.

BDD (англ. Behavior-driven Development) — процес розробки програмного забезпечення, що виник з керованої тестами розробки.

TDD (англ. Test-driven Development) — технологія розробки програмного забезпечення, яка використовує короткі ітерації розробки, що починаються з попереднього написання тестів, які визначають необхідні покращення або нові функції.

WSL (англ. Windows Subsystem for Linux) — це прошарок сумісності для запуску виконуваних файлів операційної системи Linux в середовищі Windows 10.

# ВСТУП

Серцево-судинні захворювання (далі ССЗ) являються основною причиною смерті в усьому світі. Не одна інша причина не призводить до смерті стількох людей, скільки призводять серцево-судинні захворювання. За оцінками, від ССЗ померло 17,9 мільйона чоловік у 2016 році, це складає близько 31% всіх випадків смерті в світі. Якщо вдаватись до більших подробиць, то 85% цих смертей сталося в результаті інсульту або серцевого нападу. Якщо говорити про країни з низьким або середнім рівнем прибутку, то показник смертності від серцево-судинних захворювань становить більше 75% від всіх випадків.

Люди намагаються запобігти ССЗ шляхом зменшення ризик-факторів, це може бути куріння, нездорове харчування, особливо ожиріння, схильність до вживання спиртних напоїв, відсутність фізичної активності. Але ті, хто вже мають серцево-судинні захворювання або у яких є ризик таких захворювань (а це люди з діабетом, підвищеним кров’яним тиском, гіперліпідемією), потребують раннє виявлення цієї проблеми, консультування та прийому лікарських засобів.

Взагалі, серцево-судинні захворювання є групою хвороб серця і кровоносних судин, в яку входять дуже багато захворювань: ішемічна хвороба серця - хвороба кровоносних судин, що постачають кров'ю серцевий м'яз; хвороба судин головного мозку - хвороба кровоносних судин, що постачають кров'ю мозок; хвороба периферичних артерій - хвороба кровоносних судин, що постачають кров'ю руки і ноги; ревмокардит - ураження серцевого м'яза і серцевих клапанів в результаті ревматичної атаки, що викликається стрептококовими бактеріями; вроджений порок серця - існуючі з народження деформації будови серця; тромбоз глибоких вен і емболія легень - утворення в ножних венах згустків крові, які можуть

зміщуватися і рухатися до серця і легенів. Хвороб пов’язаних з кровоносною системою дуже багато, а хворих цими захворюваннями ще більше.

# 1 ПОСТАНОВКА ЗАДАЧІ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ КАРДІОЛОГІЧНОГО WEB-ДОДАТКУ

Реалізувати автоматизоване тестування кардіологічного веб-додатку в хмарному середовищі за допомогою теоретичної літератури, наданої дипломним керівником, яке зможе контролювати працездатність програмного продукту без втручання людини, що допоможе зберегти щоденний глибокий моніторинг системи, заощадить час для мануального тестування продукту, зменшить ризик пропустити помилки, які можуть призвести до неправильної роботи та результатів продукту.

Як результат розробки системи тестування, кардіологічний web-додаток буде забезпечений постійним контролем. У відповідність з процесами або методологіями розробки ПО, під час проведення тестування створено і використовується певна кількість тестових артефактів (документи, моделі і т.д.). В цій роботі я використала такі тестові артефакти:

1. План тестування (Test Plan) - це документ для опису обсягу, ресурсу, підходу до тестування та його графік. Він складається опису з тестованих об'єктів, стратегії, розкладу, критеріїв початку і закінчення тестування, необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення. Також у тест-плані можуть бути описані ризики, що потребують планування дій у надзвичайних ситуаціях.
2. Набір тест кейсів і тестів (Test Case & Test suite) - це послідовність дій, по якій можна перевірити чи відповідає тестована функція встановленим вимогам.
3. Дефекти / Баг репорт (Bug Reports / Defects) - це документи, що описують ситуацію або послідовність дій, які призводять до

некоректної роботи об'єкта тестування, із зазначенням причин і очікуваного результату.

Мій план підготовки до автоматизації тестування складав такі етапи і його наглядно демонструє схема (Рис. 1) :

1. Аналіз вимог, на цьому етапі я розпитувала про додаток, наводила уточнюючі запитання, визначала обсяг роботи.
2. Тестове планування, на цьому етапі я робила приблизну оцінку часу, який витрачу на тестування, обирала тестові інструменті, які найбільш підходять до архітектури кардіологічного додатку.
3. Тестовий дизайн, на цьому етапі я підготувала вхідні дані для тестування, розробила тестову архітектуру.
4. Тестове виконання, на цьому етапі я програмувала створені на попередніх етапах тест-кейси, заводила баг-репорти.
5. Тестове завершення, на цьому етапі я складала тест-репорт, в якому описувала кількість багів, їх відсоткове співвідношення, в яких частинах продукту їх траплялось найбільше.

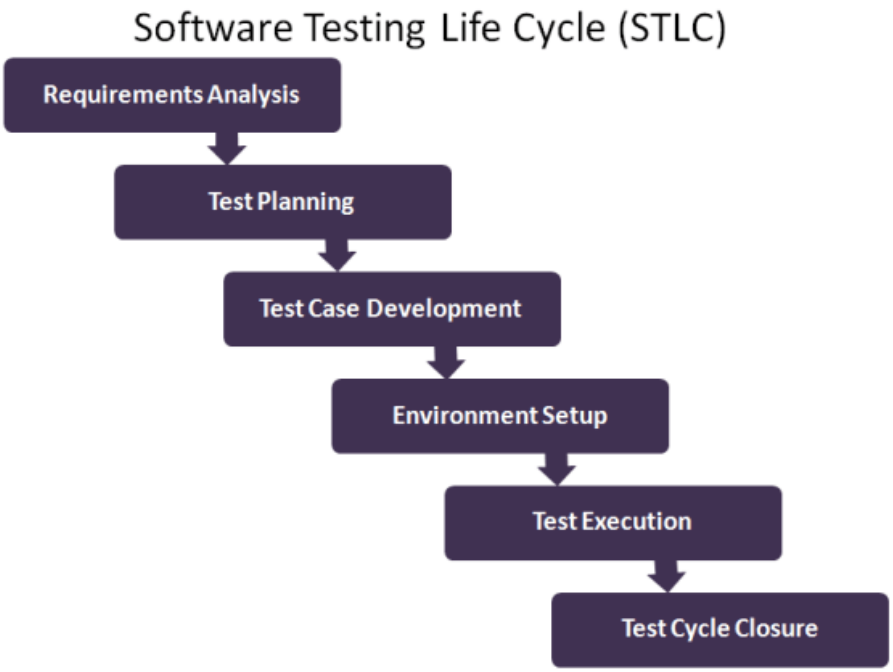


Рис. 1- Життєвий цикл тестування програмного продукту

# 2 АНАЛІЗ ПРОБЛЕМИ СТВОРЕННЯ КАРДІОЛОГІЧНИХ ПРОДУКТІВ БЕЗ ПІДТРИМКИ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

На прикладі 2020 року ми можемо бачити переповнені лікарні через COVID-19. Пацієнти з серцево-судинними факторами ризику і встановленими серцево-судинними захворюваннями представляють вразливу групу людей під час пандемії COVID-19, а пацієнти з інфарктом міокарда в контексті коронавірусу мають підвищений ризик захворюваності і смертності. Нижче на Рис. 2.1 наведено графіки смертності та дзвінків у службу порятунку з причини ССЗ до спалаху COVID-19 у місті Нью-Йорк та після спалаху. Можемо бачити, що кількість випадків дуже зросла.

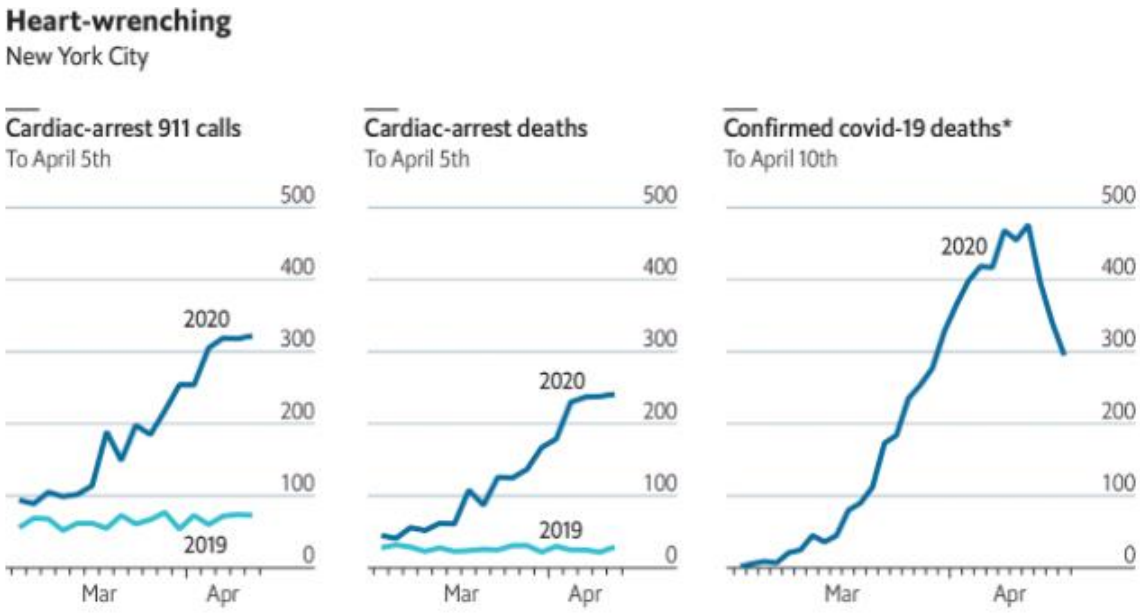


Рис 2.1 – Графіки смертності людей з ССЗ від COVID-19

Зараз ситуація, коли лікарі просто не мають часового ресурсу щоденно дивитись за показниками пацієнта, а пацієнти з серцево-судинними захворюваннями не мають змоги залишати домівку, навіть, з причини діагностики їх хвороби, є нормою. Кардіологічний web-додаток дає

можливість лікарям слідкувати за станом діяльності серця пацієнта. Завдяки щоденному моніторингу пацієнтів з захворюванням серцево-судинної системи можна запобігти розвитку важких ускладнень та зменшити загальну летальність таких пацієнтів, вчасно направити пацієнтів в спеціалізовані клініки для корекції захворювань серцево-судинної системи.

У додатку лікар може відслідкувати електрокардіограму пацієнта у першому типі біполярного відведення кінцівок (Рис.2.2) та кількість ударів серця в хвилину.

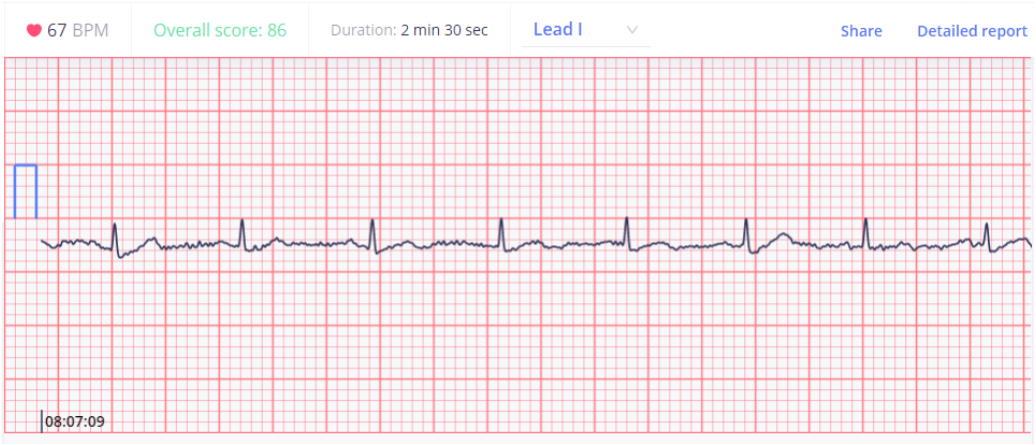


Рис 2.2 – ЕКГ пацієнта станом на 13.05.2020

У кардіологічному веб-додатку також відображається середній QRS комплекс (Рис. 2.3), який допомагає лікарю не продивлюватись усю довгу стрічку електрокардіограми, завдяки тому, що програмний продукт обчислює амплітуду зубців: P, Q, R, S, T, виводить їх середнє та малює один комплекс, виходячи з середніх данних.

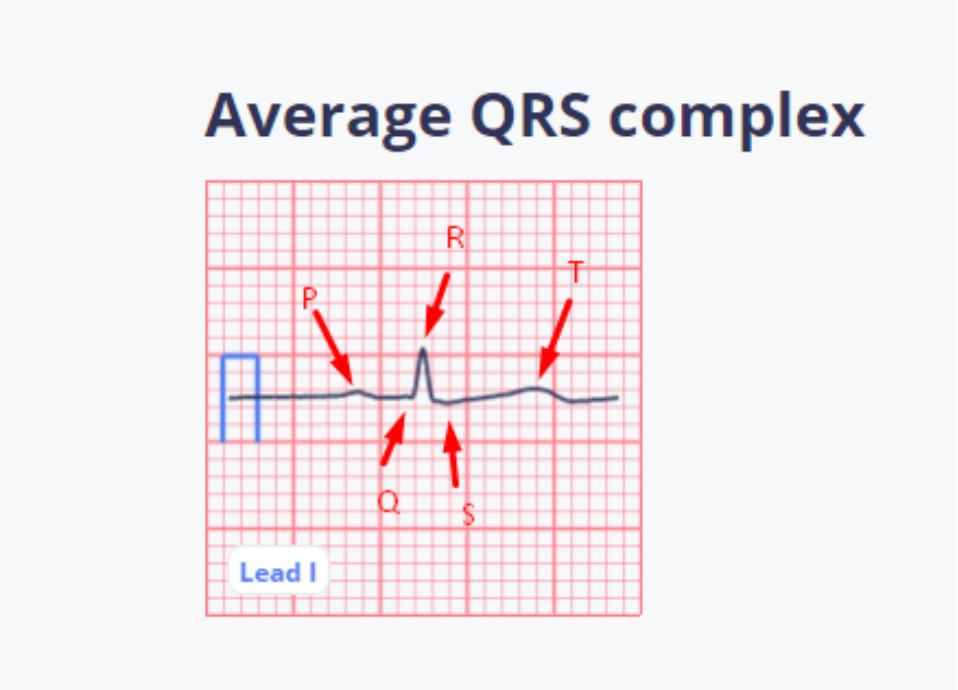


Рис 2.3 – Середній вирахований QRS комплекс з позначеними зубцями P, Q, R, S, T.

QRS комплекс є важливим у моніторингу стану серцево-судинної системи, адже патологічний зубець Q засвідчує ішемію міокарду, наприклад якщо він буде більше 1 міліметру на електрокардіограмі, то лікар дізнається про ризик інфаркту у такого пацієнта, а зникнення чи сплощення зубця T, так звана дуга Парді (Рис. 2.4), ознакою найгострішої фази інфаркту міокарду. Якщо лікар виявить на електрокардіограмі між комплексами різну відстань, то це значить, що у пацієнта неритмічний пульс, тобто захворювання аритмія, або ж на кардіограмі не буде зубця P, що визначить діагноз - мерехтлива аритмія.

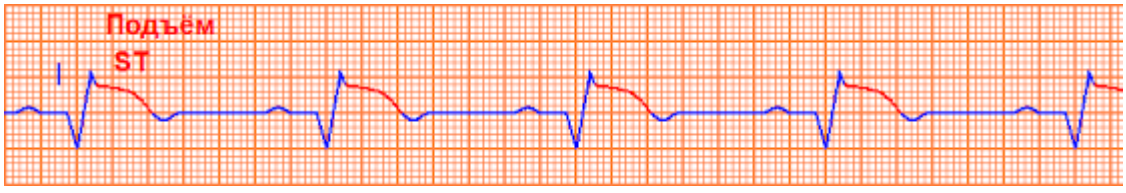


Рис 2.4 – дуга Парді.

У додатку також відображуються ЕКГ-параметри (Рис. 2.5), такі як інтервал QT (тривалість деполяризації шлуночків та реполяризація), інтервал PR (тривалість серцевого циклу шлуночків), амплітуда зубців P, Q, R, S, T, J, J40, J80, параметри варіабельності серцевого ритму.


Рис 2.5 – ЕКГ-параметри.

Всі ці дані є дуже важливими показниками загального стану пацієнта з серцево-судинним захворюванням, але дуже важливо, щоб ці дані коректно рахувались та відображались. Уявімо, що портативний пристрій для ЕКГ зчитав дані правильно, але кардіологічний веб-додаток намалював кардіограму неправильно чи порахував амплітуду неправильно, тоді у здорової людини можуть виявити ішемію міокарду чи аритмію, а у хворої в потрібний час не помітити ознаки інфаркту міокарда.

Створена система автоматизованного тестування повинна охопити усі ділянки кардіологічного web-додатку та відслідковувати його поведінку, а в разі неочікуваних результатів сповіщати про це розробників. Переваг автоматизованого тестування безліч:

1. Вони завжди виконуються однаково.
2. Немає поняття “людський фактор”.
3. Виконання набагато швидше за мануальне тестування.
4. Виконання може відбуватися в неробочі години.

5. Звіти про тестування можуть створюватися автоматично, що дає змогу відправити його на пошту, наприклад розробнику додатку.
6. Навантажувальне тестування програмного продукту може відбуватися тільки завдяки автоматизованому тестуванню.
7. За допомогою автоматизованого тестування є змога перевірити найвіддаленіші кутки додатку, які мануальний тестувальник не буде мати змогу перевірити.
8. Будь-який розробник має змогу доступитися до результатів тестування, так як вони зберігаються у хмарному середовищі.

# 3 ЗАСОБИ РОЗРОБКИ

Для написання системи автоматизованого тестування я обрала мову JavaScript та бібліотеку Nightmare. Це сучасна система тестування на базі Electron, яка дозволяє виконувати тести у віконному режимі для цілей налагодження. Nightmare знаходиться в активному розвитку і має приємний API для написання тестів. При порівнянні Nightmare з Selenium - Nightmare був майже в 3 рази швидшим, ніж Selenium, це був вирішальний фактор для визначення з інструментами тестування.

Також у ході тестування мені доводилось доступатися до мови розмітки HTML5 та до спеціальної мови стилю сторінок CSS.

## 3.1 Середовище розробки Visual Studio Code

Visual Studio Code — засіб для створення, редагування та зневадження сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки стало першим крос-платформовим продуктом у лінійці Visual Studio.

За основу для Visual Studio Code використовуються напрацювання вільного проекту Atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js.

Редактор містить вбудований зневаджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як мало енергозатратне

рішення, що дозволяє обійтися без повного інтегрованого середовища розробки. Серед підтримуваних мов і технологій: JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, DockerFile, Coffee Script, Java, HandleBars, R, Objective-C, PowerShell, Luna, Visual Basic, Markdown, JSON, HTML, CSS, LESS і SASS, Haxe.

## 3.2 Веб-інтерфейс

Веб-інтерфейс – це середовище взаємодії користувача і програми або додатку, що працює на віддаленому сервері. Частіше всього веб-інтерфейс використовується для роботи з різними онлайн сервісами, наприклад, це може бути електронна пошта чи система веб-аналітики. В окремих випадках веб-інтерфейс називається “особистим кабінетом” на деяких сайтах, але не всі особисті кабінети є інтерфейсами. Розберемо це поняття по частинам. Приставка “Веб” означає, що елемент працює віддалено від комп’ютера користувача, на локальному або інтернет-сервері. Взаємодія з сервісом при цьому відбувається через “інтерфейс” – спеціальну графічну оболочку, яка складається з кнопок, вікон, полів та інших графічних елементів.

Класичним найбільш популярним методом створення веб-інтерфейсу є використання технолії HTML з додаванням CSS та JavaScript. Проте різні реалізації HTML, CSS викликають проблеми під час розробки веб-додатків і їх подальшій підтримці. Крім того, можливість користувача налаштувати велику частину параметрів браузера, а саме розмір шрифту, колір, плагіни, може перешкоджати коректній роботі інтерфейсу. Другий менш універсальний підхід полягає у використанні технології Adobe Flash або Java-апплетів для повної або часткової реалізації користувацького інтерфейсу. Однак більшість браузерів підтримують ці технології за допомогою плагінів, тому Flash та Java-додатки можуть виконуватись з легкістю. Так як вони надають програмісту більший контроль над інтерфейсом, вони здатні

оминути багато конфліктівв конфігураціях браузерів, але використання Flash та Java реалізації може приводити до ускладнень, а саме проблем з безпекою та конфіденціальністю даних.

## 3.3 Бібліотека Nightmare

Пакет можна використовувати для тестування сайтів, для парсинга даних та для автоматизації введення даних на сайтах.

Nightmare простотий у використанні, підтримує html5, немає ніяких конфліктів із сайтами.

Клас Nightmare використовує електронний фреймворк для кожного сайту, створеного об'єктом (BrowserWindow), який використовує браузерну оболонку Chromium.

Принцип роботи у Nightmare такий:

1. Nightmare ініціалізує новий додаток electron зі стартовою сторінкою, яку необхідно обробити в подальшому.
2. Попередньо завантажуючи досліджувану сторінку, завантажуються скрипти, які дозволяють підтримувати двосторонню взаємодію програми та сторінки через серію еміттерів.
3. Nightmare надає програмісту перелік інструментів для роботи з ресурсом, дозволяючи виробляти будь-які маніпуляції з сайтами і отримувати потрібні дані.

У Nightmare є багато переваг. Код на сторонні клієнту і веб-сайту написаний на одній мові, ніяких шаблонізаторів не вимагає.

Можливість розширювати модулі через створення екшенів. Екшен може створюватись на рівні класу Nightmare чи на рівні Electron, що дає можливість використовувати Devapi Chromium.

Всі команди є ланцюгами, кожен з яких вертає promise (інтерфейс для здійснення асинхронних операцій), і дозволяє писати код як у стилі promis-ів, так і всередині асинхронних функцій або генераторів.

Відносно невелике навантаження на процесор і пам’ять, необхідно пам’ятати, що порівнювати подібний інструмент з простим GET і POST-запитом не є коректним, по швидкості та пам’яті браузерні парсери програють.

Робота Nightmare можлива в двох режимах: режим відображення браузера та режим фонового процесу.

Nightmare підтримує проксі, у ньому є можливість вмикати або вимикати відображення картинок та анімацій, підтримку webGL.

Можливо створити скрипти, що завантажуються перед виконанням програми, які дозволяють додавати на сторінку функції та бібліотеки до її завантаження.

## 3.4 Chrome DevTools

DevTools (інструменти розробника) – це програми, які дозволяють створювати, тестувати та налагоджувати програмне забезпечення. Сучасні браузери мають вбудовані інструменти розробника, що дозволяють переглянути вихідний код сайту. З їх допомогою можна переглядати й налагоджувати HTML сайту, його CSS і JavaScript. Також можна перевірити мережевий трафік, споживаний сайтом, його швидкодію та багато інших параметрів.

Як відкрити DevTools у браузері Chrome:

1. Клавіша F12.
2. Нажати одночасно клавіші Ctrl + Shift + I.
3. ПКМ за елементом сторінки, далі Переглянути код.

4. Меню браузера, далі Додаткові інструменти, далі Інструменти Розробника.

Розташовуватись консоль може знизу або збоку сторінки, також її можна показати в окремому вікні. Розглянемо по окремості кожну вкладку відкритої консолі.

У ній є 8 вкладок, які відомі з усіх даних:

1. Elements (Елементи) - містить весь html / css код сторінки і дозволяє вибрати елементи для досліджень, а також редагує їх.
2. Console (Консоль) - показує наявність / відсутність помилок / попередження у коді.
3. Sources (Джерела) - можливість виконувати операції із кодом сторінкою.
4. Network (Мережа) - відслідковує час виконання конкретних запитів і самі запити.
5. Timeline (Хронологія) - рахує час завантаження сторінки.
6. Profiles (Профілі) - дає можливість створити JavaScript, профілі CPU.
7. Resources (Ресурси) - дає можливість переглянути певні збережені дані.
8. Audits (Аудит) - проведення перевірок конкретних даних.

Панель Elements показує розмітку сторінки так само, як вона рендериться у браузері. Можна візуально змінювати наповнення сайту за допомогою зміни html / css коду в панелі елементів, в лівому віконці відображається html-документ, в правому – css (Рис. 3.1).

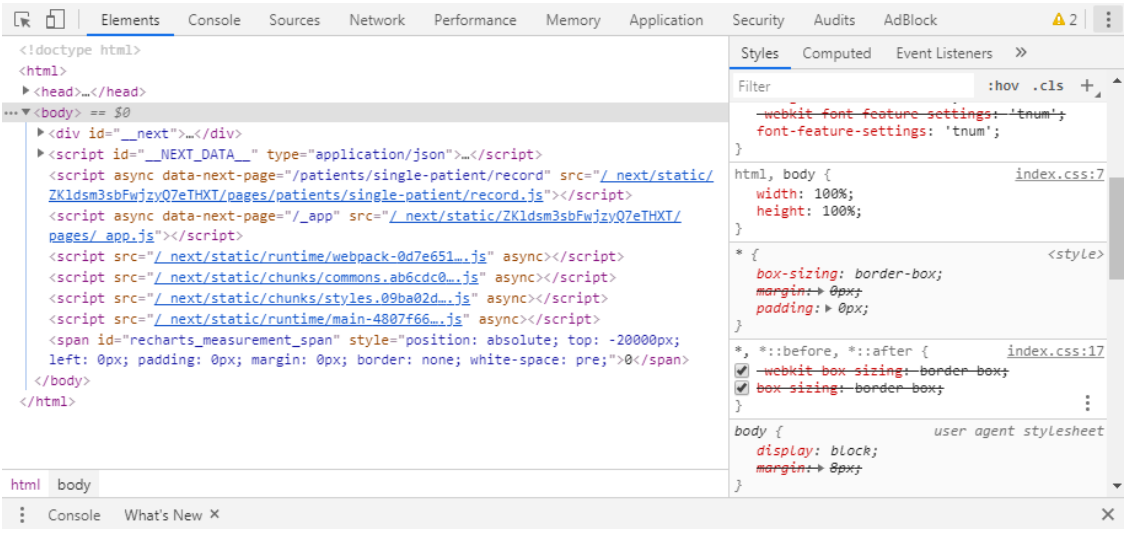


Рис 3.1 – вкладка Elements

Проводячи маніпуляції з даними можна змінити наповнення і дизайн відкритої сторінки. Наприклад, можна змінити текст у вікні, якщо редагувати його в тілі html, а також змінити шрифт сторінки помінявши його значення в поле css, але це не збереже введених даних, а допоможе просто візуально оцінити застосовані зміни. Крім цього, можна переглянути код конкретного елементу сторінки. Для цього потрібно натиснути праву кнопку миші на нього і вибрати команду «Переглянути код».

В панелі Elements є одна дуже гарна функція. Можна подивитися, як би виглядала відкрита сторінка на якомусь девайсі з іншим розширенням екрану. Натиснувши по іконці телефону зліва від вкладки Elements викликається вікно, в якому можна змінювати розмір передбачуваного екрану, таким чином емулюючи той чи інший девайс, і контролювати відображення сторінки на ньому. Виглядає це так (Рис. 3.2):

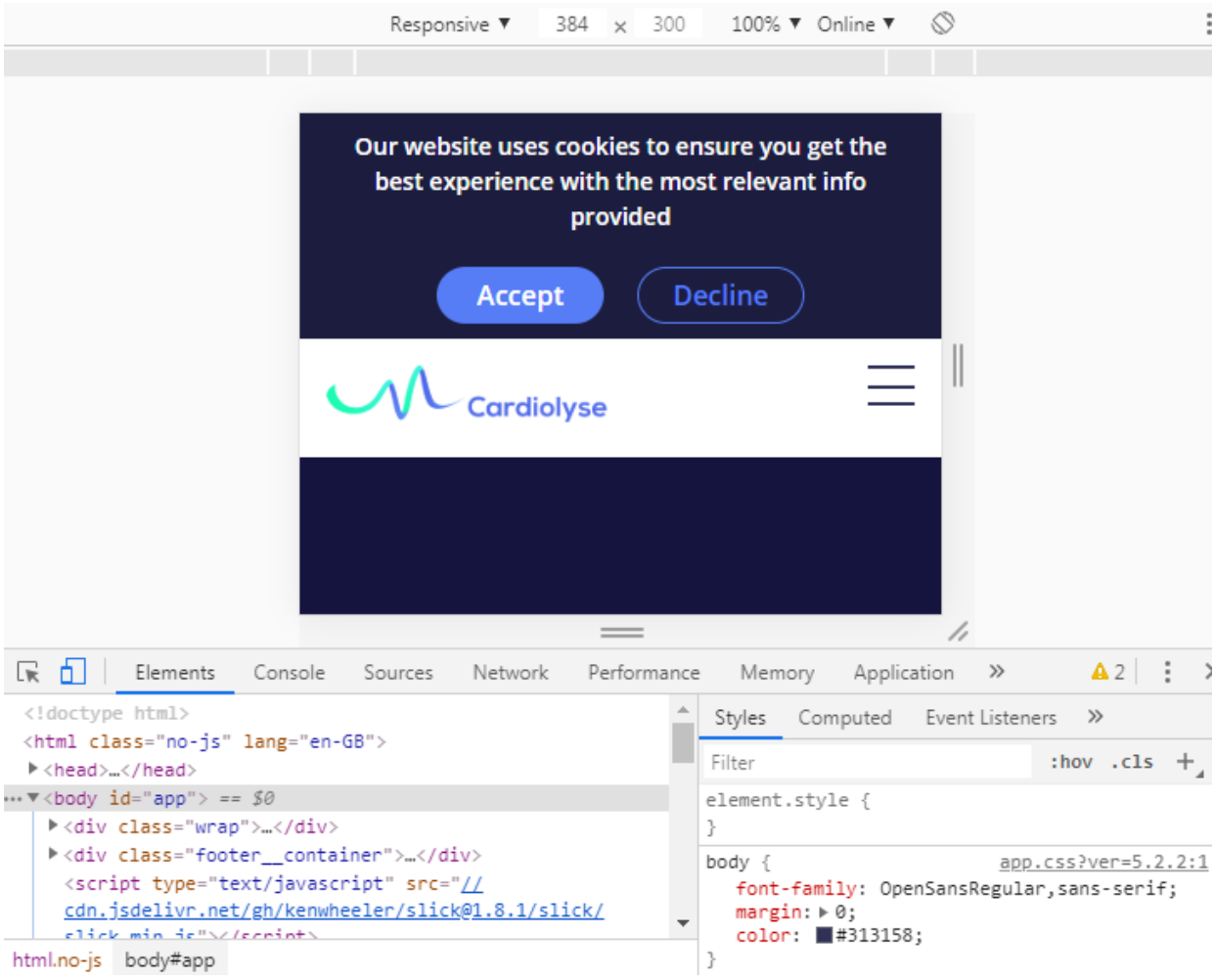


Рис 3.2 – вкладка Адаптивний екран

При натисканні на кнопку Select Model випадає дропдаун з величезним вибором девайсів. Сторінка відображена так, як якщо б це був девайс. Таким чином панель Elements можна використовувати не тільки для перегляду або редагування сторінки, але і для емуляції пристроїв відображення.

В панелі Консоль ми можемо побачити знайдені при виконанні скрипта помилки в коді. Також дана панель відображає різного роду попередження та рекомендації (Рис. 3.3). Всі виведені у вкладці повідомлення можна фільтрувати. У помилці також відображається її розташування і натиснувши на розташування ви переміститеся у вкладку Sources, де помилка буде виведена в загальній конструкції сторінки.

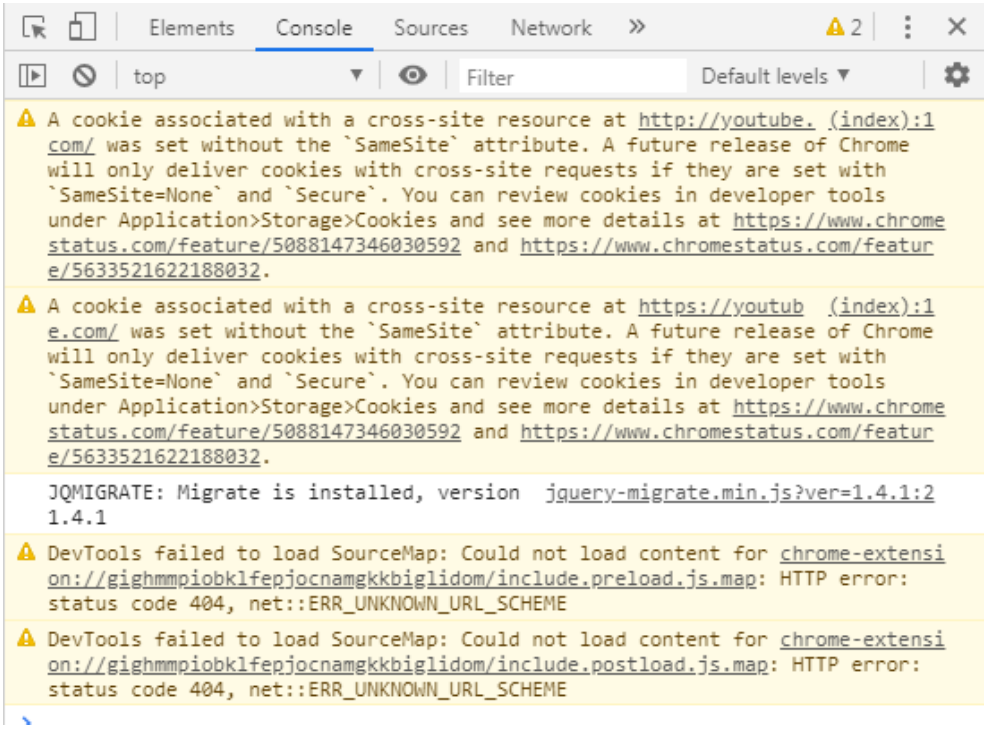


Рис. 3.3 – попередження у вкладці Console кардіологічного веб-додатку

Фільтрувати повідомлення в консолі можна по типу - помилки, попередження, інфо, стандартний вивід, повідомлення відладчика, виправлені - вибравши одну з доступних опцій консолі.

У вкладці Sources (Рис. 3.4) проводиться налагодження коду програмістами. Вона має 3 вікна (зліва направо):

1. Зона вихідних файлів. У ній знаходяться всі підключені до сторінці файли, включаючи JS / CSS.
2. Зона тексту. У ній знаходиться текст файлів.
3. Зона інформації та контролю.

У зоні вихідних файлів обирається необхідний елемент, в зоні тексту виконується безпосередньо його налагодження, а в зоні інформації та контролю можна запускати / зупиняти налагоджений код.

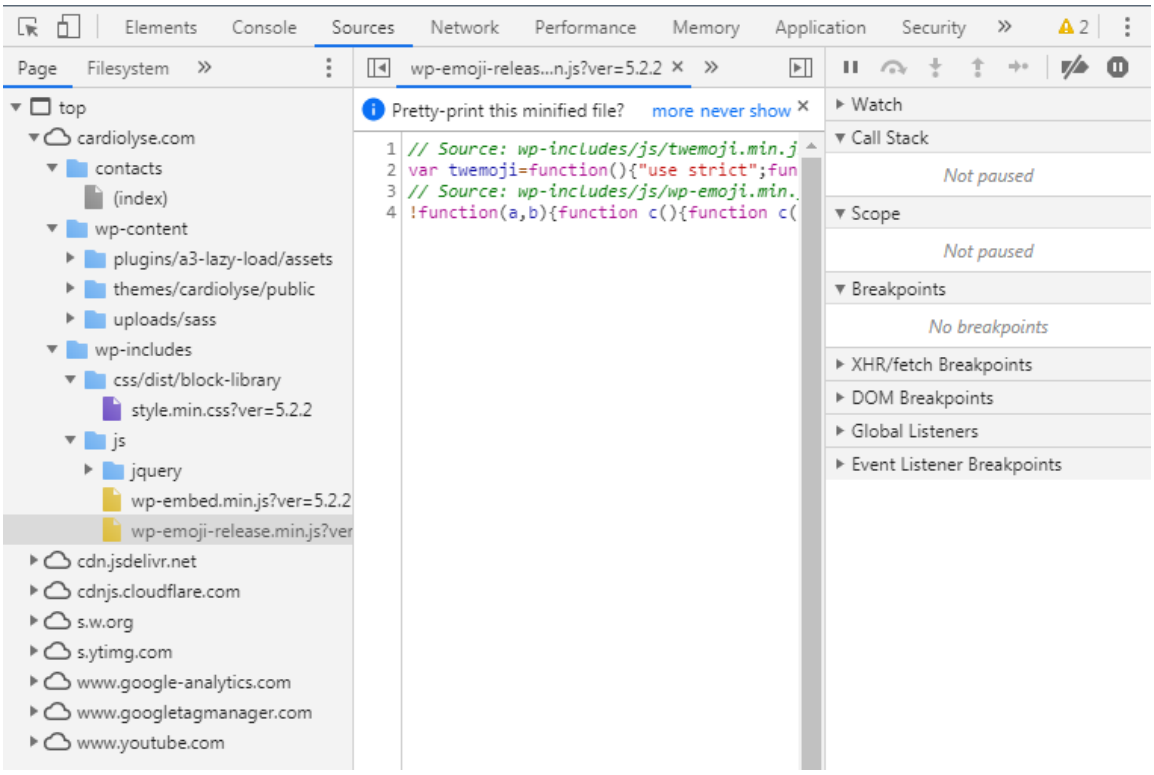


Рис. 3.4 – Вкладка Sources

Функція вкладки Network - запис мережевого журналу (Рис. 3.5). Вона дає уявлення про запитувані і завантажувані ресурси в режимі реального часу. Можна виявити, завантаження і обробку яких саме ресурсів займає більшу кількість часу, щоб згодом знати де і в чому саме можна оптимізувати сторінку.

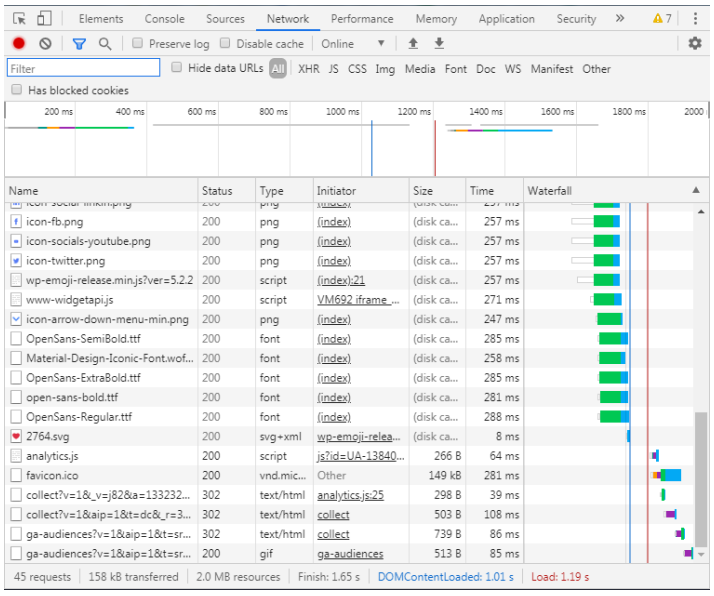


Рис. 3.5 – Мережевий журнал у вкладці Network

Також в цій вкладці в режимі Large request rows можна переглянути запити, які відправляються на сервер, а також відповіді, які приходять з нього, їх зміст і характеристики.

Вкладка Performance (Рис. 3.6) використовується при необхідності повного огляду витраченого часу. На що час був витрачен, як багато його треба було на той чи інший процес. Тут відображається абсолютно вся активність, включаючи завантаження ресурсів і виконання Javascript.

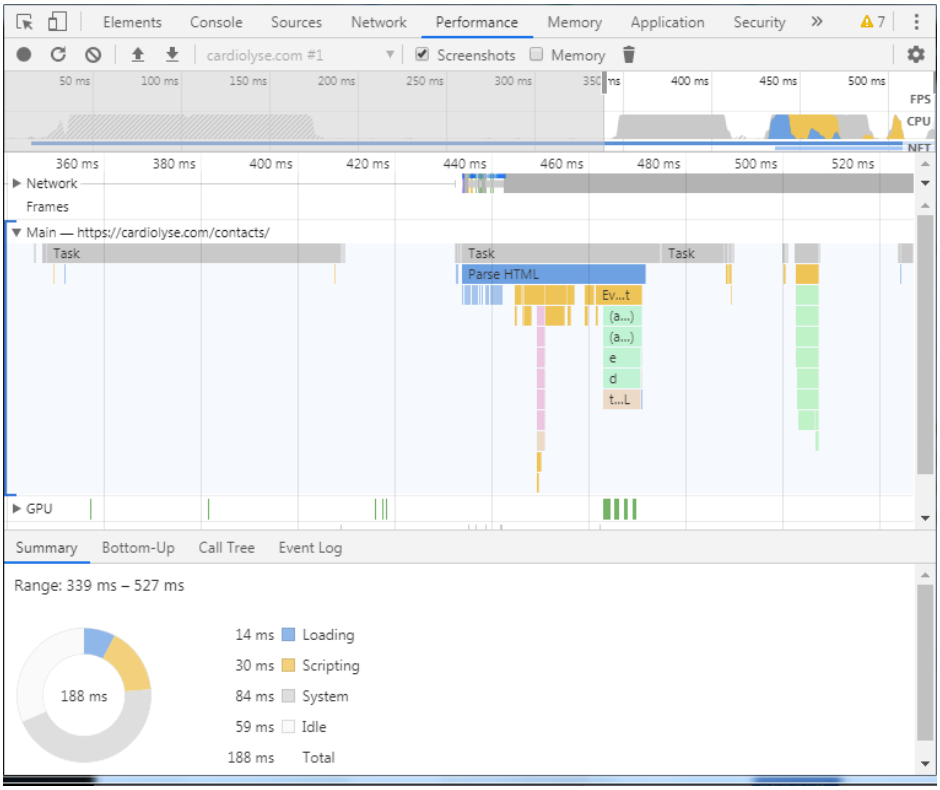


Рис. 3.6 – Вкладка Performance

Вкладка Memory (Рис. 3.7) дає можливість профілювати час виконання і використання пам'яті веб-додатком або сторінкою, таким чином допомагаючи зрозуміти де саме витрачається багато ресурсів і як можна оптимізувати код.

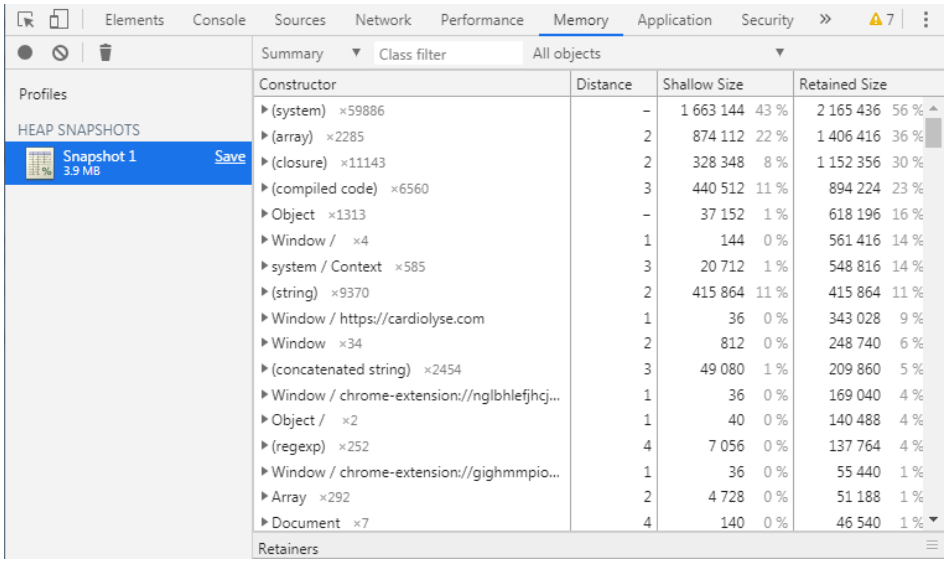


Рис. 3.7 – Вкладка Memory

CPU profiler надає інформацію про час, який витрачений на виконання Javascript. Heap profiler відображає розподіл пам'яті. JavaScript profile деталізує, куди саме був витрачений час при виконанні скриптів.

Вкладка Application призначена для дослідження завантажених елементів. Дозволяє взаємодіяти з HTML5 Database, Local Storage, Cookies (Рис. 3.8), AppCache і т.д.

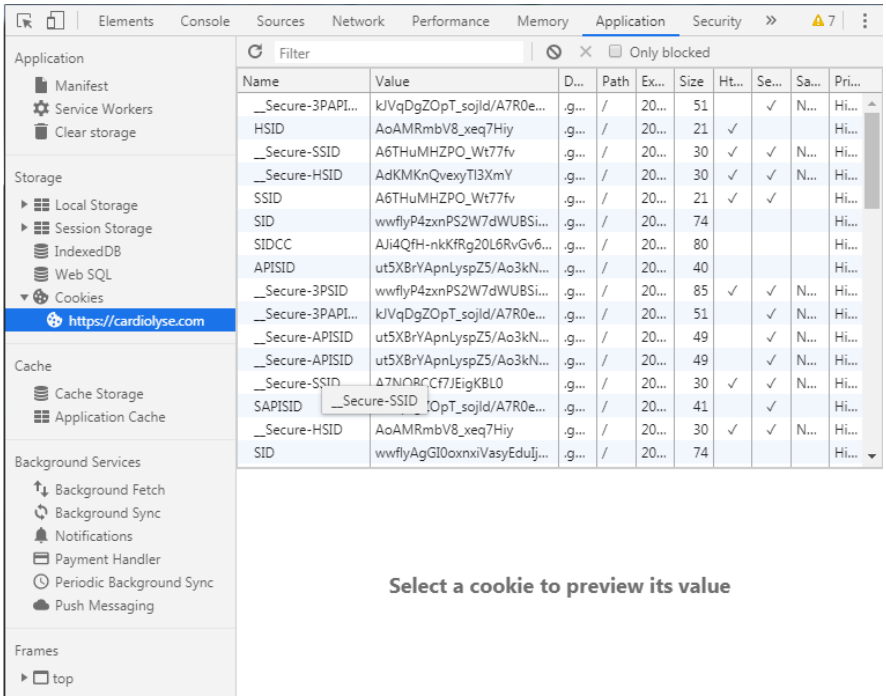


Рис. 3.8 – Перегляд cookie-файлів за допомогою вкладки Application

Вкладка Audits функціонує в якості аналізатора сторінки під час її завантаження. В результаті проведеного аудиту з'являються рекомендації щодо оптимізації завантаження сторінки, збільшення швидкості відповіді серверу. Ця вкладка містить інструмент Lighthouse. Lighthouse - це автоматизований інструмент з відкритим кодом для підвищення продуктивності, якості та коректності веб-додатків.

Під час аудиту сторінки Lighthouse проводить ряд тестів зі сторінкою, а потім формує звіт про те, наскільки це добре зробила сторінка. Якщо тест невдалий – то це індикатор того, що можна зробити для вдосконалення програми. Приклад роботи Lighthouse зображено на Рис. 3.9.

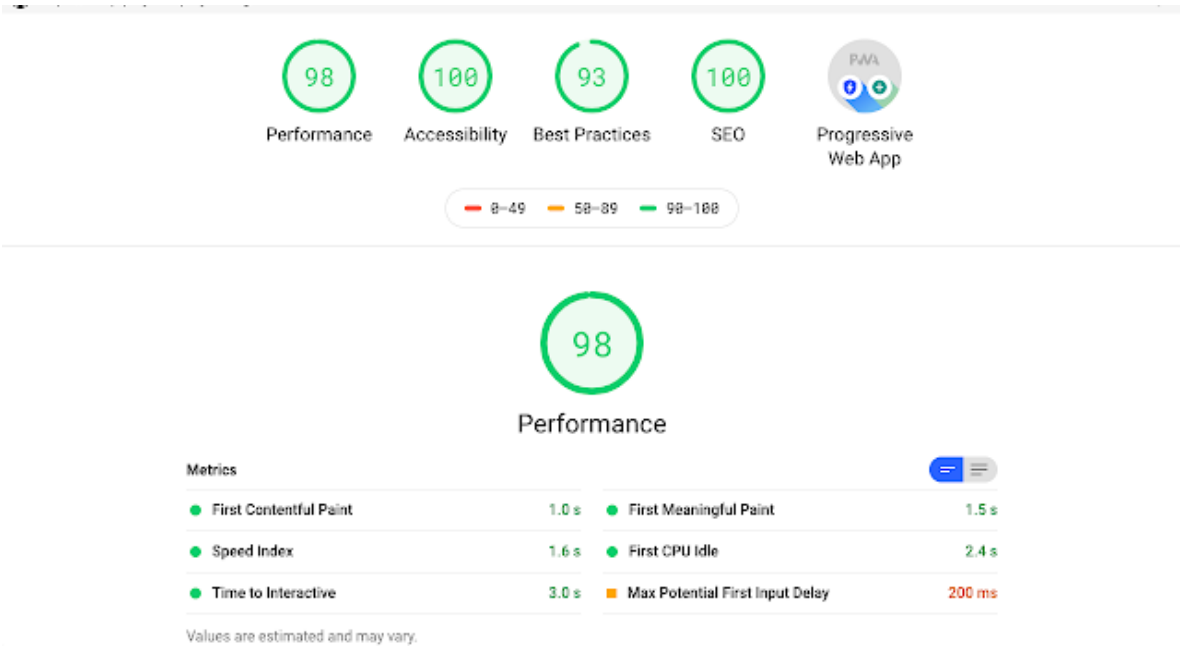


Рис. 3.9 – Результат аналізу роботи кардіологічного веб-додатку за допомогою інструментів Lighthouse

Вкладка Security показує інформацію по кожному запиту і підсвічує ті, через які сайт не отримує заповітної зеленої іконки в статусі. Крім того можна отримати наступну інформацію:

1. Про перевірку сертифіката, чи підтвердив сайт свою справжність TLS-сертифікатом.

2. TLS-з'єднання, позначає чи використовує сайт сучасні безпечні протоколи.
3. Безпеку підвантажуваних другорядних джерел.

## 3.5 Хмарна PaaS-платформа Heroku

Heroku — хмарна PaaS-платформа (Platform as a service), що підтримує ряд мов програмування. Heroku, одна з перших хмарних платформ, з'явилась в червні 2007 року і спочатку підтримувала тільки мову програмування Ruby, але на даний момент список підтримуваних мов також включає в себе Java, Node.js, Scala, Clojure, Python і PHP. На серверах Heroku використовуються операційні системи Debian або Ubuntu. Програми, що працюють на Heroku, використовують також DNS-сервер Heroku, зазвичай додатки мають доменне ім'я виду «ім'я_додатку.herokuapp.com». Для кожної програми виділяється кілька незалежних віртуальних процесів, які називаються «dynos». Вони розподілені по спеціальній віртуальній сітці («dynos grid»), яка складається з декількох серверів. Heroku також має систему контролю версій Git.

# 4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У ході реалізації автоматизованої системи тестування я створила тестові артефакти для документації етапів розробки автоматизованого тестування кардіологічного веб-додатку.

У відповідність з процесами або методологіями розробки ПО, під час проведення тестування створюється і використовується певна кількість тестових артефактів (документи, моделі і т.д.). На початку розробки я використала тест-план для опису обсягу робіт, стратегії, критеріїв початку і закінчення тестування. Для автоматизації тестування я створила тест-кейси, які допомогали в написанні програмної частини. На практиці це виглядало як: 1 тест-кейс – дискретно-реалізована одиниця програмного коду. У ході тестування, коли тест-кейси не відповідали очікуванним результатам, було створено баг-репорти. Після завершення тестування було створено тест-репорт для опису всіх тестових активностей і результатів, які були отримані після виконаної роботи за певний період часу на проекті або ж на певному тестовому рівні.

## 4.1 Тест-план

Введення

Метою складання даного Тест-плану є опис процесу тестування кардіологічного веб-додатку Cardiolyse.

Обсяг робіт

Функціонал головної сторінки Cardiolyse:

Табл. 4.1 – Опис основного функціоналу головної сторінки Cardiolyse

| Тестований функціонал | Очікувані результати | Специфіка |
|---|---|---|

| | | |
|---|---|---|
| Натискання на пункт меню About | Користувачу відкривається інформаційна сторінка кардіологічного додатку | При наведенні на пункт меню він підкреслюється синьою смужкою |
| Вхід у свій аккаунт | Користувач переходить до свого аккаунту, якщо він не виходив з системи, якщо – ні, відкривається форма входу. | При наведенні на пункт меню він підкреслюється синьою смужкою |
| Перегляд будь-якої статті у блоці Case Study | Користувач бачить статтю у повному обсязі | Стаття при натисканні на неї обрамляється блакитною рамкою |
| Натискання на будь-яку крапку у блоці Case Study | Блоки з цього розділу перемотуються вперед або назад, в залежності на яку крапку користувач нажимає | Блоки також перемотуються за рахунок стрілок зправа і зліва від розділу |
| Натискання на пункт меню Product | Користувач бачить поп-ап з пунктами меню для вибору специфікації додатку | Коли користувач наводить мишкою на любий пункт менб з поп-апу, він стає темнішим |
| Натискання на пункт меню Blog | Користувач бачить перелік статтей згрупованих по категоріям | На статті у правому верхньому кутку на білому фоні висвічується категорія |
| Натискання на пункт меню Contact us | Користувач бачить сторінку з формою для | Поля, які необхідно обов’язково заповнити, |

| | | |
|---|---|---|
| | заповнення особистих данних | позначаються зеленою зірочкою у формі |
| Натискання на кнопку Schelude a demo | Користувач бачить сторінку з формою для зворотній зв’язку | Поля, які необхідно обов’язково заповнити, позначаються синьою зірочкою у формі |
| Натискання на кнопку Read more | Користувач бачить більш детальну інформацію про ту чи іншу специфікацію додатку. | Кнопка Read more знаходиться біля блоків з специфікаціями додатку. |
| Натискання на кнопку Explore cooperation | Користувач бачить сторінку з формою для зворотній зв’язку | Поля, які необхідно обов’язково заповнити, позначаються синьою зірочкою у формі |
| Натискання на будь-яку новину у блоці News | Користувач бачить новину в повному обсязі | Знизу новини знаходиться слайдер з іншими запропонованими новинами |

Функціонал сторінки Contact us сайту Cardiolyse:

Табл. 4.2 – Опис основного функціоналу сторінки Contact us

| Тестований функціонал | Очікувані результати | Специфіка |
|---|---|---|
| Заповнення поля для імені | Користувач після початку заповнення поля ім’ям бачить як плейсхолдер зникає | Користувач не може відправити повідомлення без заповнення поля імені |

| | | |
|---|---|---|
| Заповнення поля для емейлу | Користувач після початку заповнення поля емейлом бачить як плейсхолдер зникає | Користувач не може відправити повідомлення без заповнення поля емейлу |
| Заповнення поля для номеру телефона | Користувач після початку заповнення поля номером бачить як плейсхолдер зникає | Користувач може відправити повідомлення без заповнення поля телефону |
| Заповнення поля для повідомлення | Користувач після початку заповнення поля повідомленням бачить як плейсхолдер зникає | Поле з повідомленням не розширюється, коли текст не вміщується, текст виходить з видимої зони зліва. |
| Натискання на кнопку Send | Користувач відправляє заповнені дані | При натисканні на кнопку Send і не заповненні одного з обов’язкових полів, незаповнене поле буде підсвічуватись і вимагати його заповнити |

Функціонал сторінки Live Demo сайту Cardiolyse:

Табл. 4.3 – Опис основного функціоналу сторінки Live Demo

| Тестований функціонал | Очікувані результати | Специфіка |
|---|---|---|
| Розділ Patients | Користувач бачить віртуальних пацієнтів і | Пацієнти відображаються |

| | | |
|---|---|---|
| | їх вік, як би він був зареєстрований лікарем у цьому додатку | згрупованими по першій букві ім’я та по алфавіту |
| Розділ Groups | Користувач бачить віртуальну групу пацієнтів, як би він був зареєстрований лікарем у цьому додатку | Групи відображаються згрупованими по даті їх додавання |
| Розділ Records | Користувач бачить пацієнтів, їх вік і їх відсоткове відношення показників до нормального пульсу, як би він був зареєстрований лікарем у цьому додатку | Пацієнти відображаються згрупованими по алфавіту |
| Поле для пошуку пацієнта | Шукає пацієнта за входженням символів в його ім’я | Користувач після початку заповнення поля ім’ям бачить як плейсхолдер зникає |
| Кнопка Update data | Оновлює блок з згрупованими пацієнтами | При оновленні блоку сторінка не оновлюється |
| Кнопка Settings | Користувач бачить пункт меню Вийти | Якщо вийти з демо версії, то користувач має змогу зайти назад |

Функціонал сторінки реєстрації сайту Cardiolyse:

Табл. 4.4 – Опис основного функціоналу сторінки реєстрації

| Тестований функціонал | Очікувані результати | Специфіка |
|---|---|---|
| Поп-ап вибору типу користувача | Користувач може вибрати, з якою він зареєструється | Користувач може бути в ролі лікаря або пацієнта |
| Поп-ап вибору спеціалізації лікаря | Користувач може вибрати спеціалізацію лікаря | Користувач перед цим повинен вибрати пункт Лікар |
| Поле для вводу емейлу | Користувач після початку заповнення поля емейлом бачить як плейсхолдер зникає | Користувач не може зареєструватися без заповнення поля емейлу |
| Поле для вводу паролю | Користувач після початку заповнення поля паролем бачить як плейсхолдер зникає | Користувач не може зареєструватися без заповнення поля паролю |
| Поле для підтвердження правильності написання паролю | Користувач повинен знову написати введений раніше пароль | Якщо паролі не співпадають, користувач не може зареєструватися |
| Поле для вводу імені | Користувач після початку заповнення поля ім’ям бачить як плейсхолдер зникає | Користувач не може зареєструватися без заповнення поля імені |
| Поле для вводу прізвища | Користувач після початку заповнення поля прізвищем бачить як плейсхолдер зникає | Користувач не може зареєструватися без заповнення поля прізвища |

Табл. 4.5 – Компоненти віртуального середовища

| Назва компонентів | Роль компонентів |
|---|---|
| Google Chrome | Браузер |
| Mozilla Firefox | Браузер |
| Opera | Браузер |
| Safari | Браузер |
| Microsoft Edge | Браузер |
| Яндекс Браузер | Браузер |
| Internet Explorer | Браузер |
| Jira | Система відстеження помилок |
| MS Office | Створення тест-кейсів та чек-листів |
| Visual studio code | Інтегроване середовище розробки |

Критерії якості та прийняття:

1. Зібрані всі необхідні артефакти: чек-листи, тест-кейси та баг-репорти.
2. У продукті не повинно бути блокуючих помилок, а також помилок з високим пріоритетом на момент закінчення тестування.

Критичні фактори успіху:

1. Доступ до Jira.
2. Доступ до Cardiolyse-продакшн, dev-оточенню та гілок.
3. Точні дедлайни.
4. Збиратися на заплановані зустрічі.
5. Обговорювати незрозумілі завдання вчасно.
6. Закінчити розробку і тестування всього запланованого функціоналу в терміни.

Табл. 4.6 – Оцінка ризику

| Ризики | Ймовірність | Вплив | Дії |
|---|---|---|---|
| Поломки на | Низька | Середній | Негайно підійти |

| | | | |
|---|---|---|---|
| серверах. | | | до DevOps і повідомити про збій. |
| Упущення різних видів помилок Junior-тестувальниками. | Висока | Високий | Компетентний фахівець допомагає вирішити проблему в разі, якщо вона термінова. |
| Лікарняні або відпустки. | Висока | Середній | Project Manager бере в облік Man-day кожного співробітника. |

Табл. 4.7 – Тестувальники

| Роль | Ім’я | Обов'язки |
|---|---|---|
| QA Engineer | Анна | Розробка тест-кейсів, чек-листів, написання баг-репортів. |

Табл. 4.8 – Перелік тест-обладнання

| Роль | Девайс | Конфігурація програмного забезпечення |
|---|---|---|
| Клієнт | IPhone 6s | iOS 12.3.1 |
| Клієнт | Laptop Dell | Windows 7 64 bit |

Табл. 4.9 – Перелік компонентів віртуального середовища

| Інструмент | Опис |
|---|---|
| Jira | Система відстеження помилок |

| Confluence | Документація |
|---|---|
| MS Office Excel | Стоверння чек-листів |
| MS Office Word | Створення тест-кейсів |

Табл. 4.10 – Перелік тестової документації

| Назва | Частота | Зберігання |
|---|---|---|
| Тест-план | 1 раз до початку тестування | Google disk |
| Чек-лист | до початку тестування | Google disk |
| Тест-кейс | до початку тестування | Google disk |
| Баг-репорт | При виявленні помилки | Jira |
| Тест-репорт | 1 раз після закінчення тестування | Google disk |

Тестова стратегія

Команда тестування може призупинити дії тестування, якщо:

1. Помилка в функції, яка перешкоджає її перевірці.
2. Виникла серйозна проблема, яка не дозволяє продовжити тестування.

Табл. 4.11 – Графік тестування

| Діяльність | Початок | Кінець |
|---|---|---|
| Створення тест-плану | 13.04.2020 | 20.04.2020 |
| Написання тест-кейсів | 20.04.2020 | 27.04.2020 |
| Програмування тест-кейсів | 27.04.2020 | 04.05.2020 |
| Тестування та заведення баг-репортів | 04.05.2020 | 11.05.2020 |
| Регресійне тестування та | 11.05.2020 | 14.05.2020 |

| | | |
|---|---|---|
| написання тест-репорту | | |

## 4.2 Тест-кейси

У цьому пункті описані тест-кейси, тобто артефакти, що описують сукупність кроків, конкретних умов та параметрів, необхідних для перевірки реалізації функціоналу, що тестується. Згідно цього задокументованого артефакту створювались тести за допомогою фреймворку Nightmare.

Табл. 4.12 – Тест-кейси на функціонал інформаційної сторінки пацієнта у кардіологічному веб-додатку.

| Тестований функціонал | Очікувані результати |
|---|---|
| Користувач натискає на пацієнта з списку всіх пацієнтів. | З'являється сторінка з блоком загальної інформації про пацієнта та блоком інформації з щоденної електрокардіограми пацієнта. |
| Користувач натискає на пацієнта з списку всіх пацієнтів. | У блоці присутні BPM ( кількість ударів серця за хвилину), загальний показник стану здоров'я, тривалість знімання електрокардіограми, кнопка вибору біполярного відведення кінцівок, кнопка для змоги поділитися з іншими лікарями інформацією пацієнта у додатку, кнопка показу детального репорту, електрокардіограма, середній комплекс QRS, блок з ЕКГ-параметрами, блок кодів Міннесоти (стандартів та |

| | |
|---|---|
| | класифікації захворювань), блок амплітуди кожного зубця електрокардіограми, блок Синдромних кодів (стандартів та класифікації захворювань), блок параметрів варіабельності серцевого ритму, комплексне резюме щоденного стану пацієнта. |
| Користувач натискає на кнопку вибору біполярного відведення кінцівок. | З'являється поп-ап вибору біполярного відведення кінцівок з 3 типів. |
| Користувач натискає на будь-який тип біполярного відведення кінцівок. | Електрокардіограма, середній комплекс QRS та блок амплітуди кожного зубця електрокардіограми відображається в залежності до типу біполярного відведення кінцівок. |
| Користувач пролистує сторінку до блоку з ЕКГ-параметрами. | Блок ЕКГ-параметрів містить інтервал QT (тривалість деполяризації шлуночків та реполяризації), інтервал коригованого QTc, інтервал коригованого QTcF (по формулі лікаря Фрідерісії), інтервал RR: тривалість серцевого циклу шлуночків (показник частоти шлуночків), тривалість QRS: тривалість деполяризації м'язів шлуночків, вісь QRS, яка являє |

| | |
|---|---|
| | собою головний вектор активації шлуночків, який є загальним напрямком електричної активності. |
| Користувач пролистує сторінку до блоку амплітуди кожного зубця електрокардіограми. | Блок амплітуди містить значення типу біполярного відведення кінцівок та таких зубців: P, Q, R, S, T, J, J40, J80. |
| Користувач пролистує сторінку до блоку параметрів варіабельності серцевого ритму та натискає на праву стрілочку. | З'являється блок з діаграмою інтервалу між двома серцебиттями (R зубця в комплексі QRS / ЕКГ). |
| Користувач натискає на праву стрілочку у блоці з діаграмою інтервалу між двома серцебиттями. | З'являється блок з частотними параметрами такими як: VLF (дуже низька частота), LF (низька частота), HF (висока частота). |
| Користувач натискає на ліву стрілочку у блоці з діаграмою інтервалу між двома серцебиттями. | З'являється блок параметрів варіабельності серцевого ритму. |
| Користувач натискає на ліву стрілочку у блоці з частотними параметрами. | З'являється блок з діаграмою інтервалу між двома серцебиттями (R зубця в комплексі QRS / ЕКГ). |
| Користувач наводить на блок Overall wellbeing Info(Загальне самопочуття) курсором. | З'являється поп-ап Overall wellbeing Info(Загальне самопочуття) з описом стадій самопочуття в залежності від відсотку. |
| Користувач наводить на блок Myocardial score(Оцінка міокарда) курсором. | З'являється поп-ап Myocardial score(Оцінка міокарда) з описом стану міокарду в залежності від відсотку. |

| | |
|---|---|
| Користувач натискає на кнопку скачати PDF. | З'являється вікно друку з сформованним звітом щоденного стану пацієнта. |
| Користувач натискає на кнопку скачати Excel. | Файл Excel автоматично формується та завантажується до ПК. |
| Користувач натискає на кнопку скачати RAW. | Файл з розширенням edf автоматично формується та завантажується до ПК. |
| Користувач натискає запис з списку щоденних електрокардіограм. | Блок щоденної інформації пацієнта змінює інформацію згідно дати, яку вибрав користувач. |

Табл. 4.13 – Тест-кейси на функціонал авторизації та реєстрації кардіологічного веб-додатку.

| Тестований функціонал | Очікувані результати |
|---|---|
| Користувач натискає на кнопку входу у власний аккаунт під назвою "Log in". | У користувача з'являється сторінка входу у власний аккаунт з адресою "https://app.cardiolyse.com/sign-in" з центральним блоком "Sign in", який містить поля "Email" та "Password", неактивну кнопку "Sign in", у правому верхньому кутку кнопка "Sign up" для реєстрації та у лівій частині сторінки кнопка "Live demo", яка дає можливість користування програмним продуктом, поки користувач ще не має аккаунта. |

| | |
|---|---|
| Користувач натискає поле "Email", нічого не вводить в це поле, натискає поле "Password". | У користувача рамка поля "Email" стає червоною, під полем з'являється напис "Required!", що свідчить користувачу про необхідність введення електронної адреси, кнопка "Sign in" є неактивною, так як користувач ще не вводив електронну адресу і пароль. |
| Користувач натискає поле "Email", вводить свою електронну адресу, яка відповідає шаблону : хоча б 1 символ латиницею, далі знак @, далі хоча б 1 символ латиницею, далі крапка, далі хоча б 2 символи латиницею. | У користувача рамка поля "Email" стає синьою, поряд з введенною електроною адресою з'являється зелений круг з білою галочкою, що свідчить, що електронна адреса пройшла верифікацію, кнопка "Sign in" є неактивною, так як користувач ще не вводив пароль. |
| Користувач натискає поле "Email", вводить електронну адресу, яка не відповідає шаблону : хоча б 1 символ латиницею, далі знак @, далі хоча б 1 символ латиницею, далі крапка, далі хоча б 2 символи латиницею. | У користувача рамка поля "Email" стає червоною, під полем з'являється напис "Invalid email address!", що свідчить користувачу про неправильність написання електронної адреси, кнопка "Sign in" є неактивною (так як користувач ще не вводив пароль та його електронна адреса не відповідає шаблону вірно введеної електронної адреси. |

| | |
|---|---|
| Користувач натискає поле "Email", вводить свою електронну адресу, яка відповідає шаблону : хоча б 1 символ латиницею, далі знак @, далі хоча б 1 символ латиницею, далі крапка, далі хоча б 2 символи латиницею та яка вже зареєстрована. Потім користувач натискає поле "Password", вводить 8 символів латиницею, які відповідають зареєстрованій пошті. | Користувач входить в свій обліковий запис. |
| Користувач нажимає кнопку "Sign up" (зареєструватися). | З'являється поп-ап з вибором ролі у додатку: пацієнт або лікар та неактивна кнопка "Continue" (продовжити). |
| Користувач обирає роль пацієнта у поп-апі вибору ролі в додатку. | Кнопка "Continue" стає активною для натискання. |
| Користувач після вибору ролі пацієнта натискає на кнопку "Continue". | З'являється форма вводу своїх даних з полями для вводу електронної адреси, ім'я, фамілії, паролю та підтвердження паролю та неактивною кнопкою "Sign up" (зареєструватися), форма має посилання https://app.cardiolyse.com/sign-up. |
| Користувач обирає роль лікаря у поп-апі вибору ролі в додатку. | З'являється поп-ап з вибором спеціалізації лікаря, яка за замовчуванням стоїть на спеціалізації сімейного лікаря, також |

| | |
|---|---|
| | є вибір з кардіолога, спортивного лікаря та дієтолога та неактивна кнопка "Continue" (продовжити). |
| Користувач обирає спеціалізацію лікаря у поп-апі вибору спеціалізації лікаря в додатку. | Кнопка "Continue" стає активною для натискання. |
| Користувач після вибору спеціалізації лікаря натискає на кнопку "Continue". | З'являється форма вводу своїх даних з полями для вводу електронної адреси, ім'я, фамілії, паролю та підтвердження паролю та неактивною кнопкою "Sign up " (зареєструватися), форма має посилання https://app.cardiolyse.com/sign-up. |
| Користувач натискає поле "Email", нічого не вводить в це поле, натискає на інше будь-яке поле. | У користувача рамка поля "Email" стає червоною, під полем з'являється напис "Required!", що свідчить користувачу про необхідність введення електронної адреси, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Email", вводить свою електронну адресу, яка відповідає шаблону : хоча б 1 символ латиницею, далі знак @, далі хоча б 1 символ латиницею, далі крапка, далі хоча б 2 символи латиницею. | У користувача рамка поля "Email" стає синьою, поряд з введенною електроною адресою з'являється зелений круг з білою галочкою, що свідчить, що електронна адреса пройшла верифікацію, кнопка "Sign up" є неактивною. |

| | |
|---|---|
| Користувач натискає поле "Email", вводить електронну адресу, яка не відповідає шаблону : хоча б 1 символ латиницею, далі знак @, далі хоча б 1 символ латиницею, далі крапка, далі хоча б 2 символи латиницею. | У користувача рамка поля "Email" стає червоною, під полем з'являється напис "Invalid email address!", що свідчить користувачу про неправильність написання електронної адреси, кнопка "Sign un" є неактивною. |
| Користувач натискає поле "Name", нічого не вводить в це поле, натискає на інше будь-яке поле. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "First name is required!", що свідчить користувачу про необхідність введення ім'я, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Name", вводить менше двох літер латиницею. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name is too short", що свідчить користувачу про те, що ім'я закоротке, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Name", вводить цифри. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name can include only letters", що свідчить користувачу про те, що ім'я має містити тільки літери латиницею, кнопка "Sign up" є неактивною. |

| | |
|---|---|
| Користувач натискає поле "Name", вводить літери кирилицею. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name can include only letters", що свідчить користувачу про те, що ім'я має містити тільки літери латиницею, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Name", вводить цифри і літери латиницею | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name can include only letters", що свідчить користувачу про те, що ім'я має містити тільки літери латиницею, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Name", вводить більше двох літер латиницею. | У користувача рамка поля "Name" стає синьою, поряд з введенним ім'ям з'являється зелений круг з білою галочкою, що свідчить, що таке ім'я підходить для реєстрації, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Name", вводить більше 20 літер латиницею. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name is too long", що свідчить користувачу про те, що ім'я занадто довге, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Surname", нічого не вводить в це поле, натискає на інше будь-яке поле. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Second name is required!", що |

| | |
|---|---|
| | свідчить користувачу про необхідність введення прізвища, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Surname", вводить менше двох літер латиницею. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname is too short", що свідчить користувачу про те, що прізвище закоротке, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Surname", вводить цифри. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname can include only letters", що свідчить користувачу про те, що прізвище має містити тільки літери латиницею, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Surname", вводить літери кирилицею. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname can include only letters", що свідчить користувачу про те, що прізвище має містити тільки літери латиницею, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Surname", вводить цифри і літери латиницею. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname can include only letters", що свідчить користувачу про те, що прізвище має містити тільки літери латиницею, кнопка "Sign up" є |

| | |
|---|---|
| | неактивною. |
| Користувач натискає поле "Surname", вводить більше двох літер латиницею. | У користувача рамка поля "Surname" стає синьою, поряд з введенним прізвищем з'являється зелений круг з білою галочкою, що свідчить, що таке прізвище підходить для реєстрації, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Surname", вводить більше 20 літер латиницею. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname is too long", що свідчить користувачу про те, що прізвище занадто довге, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Password", нічого не вводить в це поле, натискає на інше будь-яке поле. | У користувача рамка поля "Password" стає червоною, під полем з'являється напис "Password is required!", що свідчить користувачу про необхідність введення пароля, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Password", вводить будь-яку комбінацію з латиничних, кириличних літер та цифр, але яка за довжиною менше 8 символів. | У користувача рамка поля "Password" стає червоною, під полем з'являється напис "Password is too short", що свідчить користувачу про те, що пароль занадто короткий, сам пароль недоступний для відображення, кнопка "Sign up" є неактивною. |

| | |
|---|---|
| Користувач натискає поле "Password", вводить будь-яку комбінацію з латиничних, кириличних літер та цифр, але яка за довжиною більше 8 символів та менше 25 символів. | У користувача рамка поля "Password" стає синьою, поряд з введенним паролем з'являється зелений круг з білою галочкою, що свідчить користувачу про те, що такий пароль підходить для реєстрації, сам пароль недоступний для відображення, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Password", вводить будь-яку комбінацію з латиничних, кириличних літер та цифр, але яка за довжиною більше 25 символів. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Password is too long", що свідчить користувачу про те, що пароль занадто довгий, сам пароль недоступний для відображення, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Confirm password", нічого не вводить в це поле, натискає на інше будь-яке поле. | У користувача рамка поля "Password" стає червоною, під полем з'являється напис "Confirm your new password!", що свідчить користувачу про необхідність повторити введення пароля, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Confirm password", вводить будь-яку комбінацію з латиничних, кириличних літер та цифр, яка не співпадає з введенним паролем у поле "Password". | У користувача рамка поля "Confirm password" стає червоною, під полем з'являється напис "Passwords do not match", що свідчить користувачу про те, що пароль не співпадає з введенним паролем у поле "Password", сам пароль недоступний |

| | |
|---|---|
| | для відображення, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Confirm password", вводить будь-яку комбінацію з латиничних, кириличних літер та цифр, яка співпадає з введенним паролем у поле "Password". | У користувача рамка поля "Confirm password" стає синьою, поряд з введенним прізвищем з'являється зелений круг з білою галочкою, що свідчить, що пароль співпадає з введенним паролем у поле "Password", сам пароль недоступний для відображення, кнопка "Sign up" є неактивною. |
| Користувач натискає поле "Email", вводить свою електронну адресу, яка відповідає шаблону : хоча б 1 символ латиницею, далі знак @, далі хоча б 1 символ латиницею, далі крапка, далі хоча б 2 символи латиницею, вводить ім'я та прізвище, які містять більше двох та менше двадцяти латинських літер, вводить пароль, який містить будь-яку комбінацію з латиничних, кириличних літер та цифр у межах від 8 до 25 символів та вірно вводить повторно пароль, що співпадає з попереднім. | Кнопка "Sign up" (реєстрації) стає активною. |

Табл. 4.14 – Тест-кейси на функціонал додавання лікарем нового пацієнта у базу кардіологічного веб-додатку.

| | |
|---|---|
| Користувач натискає кнопку додати пацієнта. | З'являється поп-ап Додавання пацієнта, який містить кнопку вибору статі пацієнта, поле для введення віку пацієнта кнопку вибору зв'язку, поле для введення імені, поле для введення фамілії, кнопка для введення додаткових полів. |
| Користувач натискає кнопку Male(Чоловіча). | Кнопка Male зафарбовується синім кольором. |
| Користувач натискає кнопку Female(Жіноча). | Кнопка Female зафарбовується синім кольором. |
| Користува натискає на поле, нічого не вводить, натискає на будь-яке інше поле. | Поле для введення віку пацієнта підсвічується червоним та має напис Required, що говорить користувачу про необхідність введення віку пацієнта. |
| Користувач натискає на поле та вводить букву "e". | Букви не вводяться у поле Вік. |
| Користувач вводить будь-яке число у проміжку від 1 до 120. | Число коректно відображається у полі Вік, рамка поля стає голубою, біля віку з'являється зелений чек-бокс. |
| Користувач вводить будь-яке число поза проміжком від 1 до 120. | Поле для введення віку пацієнта підсвічується червоним та має напис "Please enter value between 1 and 120", що говорить користувачу про необхідність введення віку в межах від 1 до 120. |
| Користувач натискає на "Email". | Кнопка "Email" стає синьою, під кнопкою з'являється поле для введення електронної пошти. |

| | |
|---|---|
| Користувач натискає на "Phone". | Кнопка "Phone" стає синьою, під кнопкою з'являється кнопка з вибором коду регіону та поле для введення номеру телефону. |
| Користувач нічого не вводить до поля "Email" та натискає на будь-яке інше поле. | Поле "Email" підсвічується червоним та має напис Required, що свідчить користувачу про необхідність введення поштової адреси. |
| Користувач нічого не вводить до поля "Phone" та натискає на будь-яке інше поле. | Поле "Phone" підсвічується червоним та має напис "Please enter valid phone number", що свідчить користувачу про необхідність введення поштової адреси. |
| Користувач вводить в поле "Phone" цифри у кількості від 9 до 13. | Рамка поля "Phone" стає голубою, біля номеру телефона з'являється зелений чек-бокс. |
| Користувач вводить в поле "Phone" цифри у кількість яких не впадає у проміжок від 9 до 13. | Поле "Phone" підсвічується червоним та має напис "Please enter valid phone number", що свідчить користувачу про необхідність введення коректного номеру телефона. |
| Користувач натискає поле "Email", вводить електронну адресу, яка відповідає шаблону : хоча б 1 символ латиницею, далі знак @, далі хоча б 1 символ латиницею, далі крапка, далі хоча б 2 символи латиницею. | Рамка поля "Email" стає голубою, біля електроної пошти з'являється зелений чек-бокс. |

| | |
|---|---|
| Користувач натискає поле "Email", вводить електронну адресу, яка не відповідає шаблону : хоча б 1 символ латиницею, далі знак @, далі хоча б 1 символ латиницею, далі крапка, далі хоча б 2 символи латиницею. | Поле "Email" підсвічується червоним та має напис "Invalid email address!", що свідчить користувачу про необхідність введення коректної поштової адреси. |
| Користувач натискає поле "Name", нічого не вводить в це поле, натискає на інше будь-яке поле. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "First name is required!", що свідчить користувачу про необхідність введення ім'я. |
| Користувач натискає поле "Name", вводить менше двох літер латиницею. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name is too short", що свідчить користувачу про те, що ім'я закоротке. |
| Користувач натискає поле "Name", вводить цифри. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name can include only letters", що свідчить користувачу про те, що ім'я має містити тільки літери латиницею. |
| Користувач натискає поле "Name", вводить літери кирилицею. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name can include only letters", що свідчить користувачу про те, що ім'я має містити тільки літери латиницею. |
| Користувач натискає поле "Name", вводить цифри і літери | У користувача рамка поля "Name" стає червоною, під полем з'являється напис |

| | |
|---|---|
| латиницею. | "Name can include only letters", що свідчить користувачу про те, що ім'я має містити тільки літери латиницею. |
| Користувач натискає поле "Name", вводить більше двох літер латиницею. | У користувача рамка поля "Name" стає синьою, поряд з введенним ім'ям з'являється зелений круг з білою галочкою, що свідчить, що таке ім'я підходить для реєстрації. |
| Користувач натискає поле "Name", вводить більше 20 літер латиницею. | У користувача рамка поля "Name" стає червоною, під полем з'являється напис "Name is too long", що свідчить користувачу про те, що ім'я занадто довге. |
| Користувач натискає поле "Surname", нічого не вводить в це поле, натискає на інше будь-яке поле. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Second name is required!", що свідчить користувачу про необхідність введення прізвища. |
| Користувач натискає поле "Surname", вводить менше двох літер латиницею. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname is too short", що свідчить користувачу про те, що прізвище закоротке. |
| Користувач натискає поле "Surname", вводить цифри. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname can include only letters", що свідчить користувачу про те, що прізвище має містити тільки літери латиницею. |
| Користувач натискає поле "Surname", вводить літери | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис |

| | |
|---|---|
| кирилицею. | "Surname can include only letters", що свідчить користувачу про те, що прізвище має містити тільки літери латиницею. |
| Користувач натискає поле "Surname", вводить цифри і літери латиницею. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname can include only letters", що свідчить користувачу про те, що прізвище має містити тільки літери латиницею. |
| Користувач натискає поле "Surname", вводить більше двох літер латиницею. | У користувача рамка поля "Surname" стає синьою, поряд з введенним прізвищем з'являється зелений круг з білою галочкою, що свідчить, що таке прізвище підходить для реєстрації. |
| Користувач натискає поле "Surname", вводить більше 20 літер латиницею. | У користувача рамка поля "Surname" стає червоною, під полем з'являється напис "Surname is too long", що свідчить користувачу про те, що прізвище занадто довге. |
| Користувач вводить будь-яку послідовність символів у поле "Patient ID" до 20 символів. | Введена послідовність символів відображується коректно. |
| Користувач вводить будь-яку послідовність символів у поле "Patient ID" більше 20 символів. | У користувача рамка поля "Patient ID" стає червоною. |
| Користувач вводить будь-яку послідовність символів у поле "Add note". | Введена послідовність символів відображується коректно. |
| Користувач рухає перемикач вправо на слайдері "Weight" | Значення ваги збільшується. |

| | |
|---|---|
| (Вага). | |
| Користувач рухає перемикач вліво на слайдері "Weight" (Вага). | Значення ваги зменшується. |
| Користувач рухає перемикач вправо на слайдері "Height" (Ріст). | Значення росту збільшується. |
| Користувач рухає перемикач вліво на слайдері "Height" (Ріст). | Значення росту зменшується. |
| Користувач заповнює коректно вісі поля та нажимає на кнопку "Add new patient". | Новий пацієнт додан до списку пацієнтів. |

## 4.3 Структура програмного забезпечення

Для реалізації автоматизованого тестування кардіологічного веб-додатку було розроблено систему авто-тестів, яка була розбита на тестові набори (Рис. 4.15), кожен з яких відповідав окремому функціоналу програми:

1. Авторизація.
2. Реєстраційна форма.
3. Реєстрація ролі.
4. Інформаційна сторінка пацієнта.

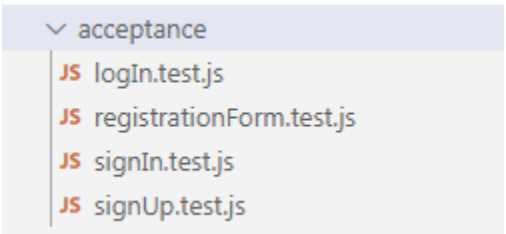


Рис. 4.15 – Перелік тестових наборів розроблених авто-тестів.

Структура програми скаладається з багатьох модулів, головні з яких: URL, Chai, Nightmare, Open.

Модуль URL надає утиліти для вирішення та розбору URL-адрес, тобто розбиває веб-адресу на читабельні частини (Рис. 4.16).

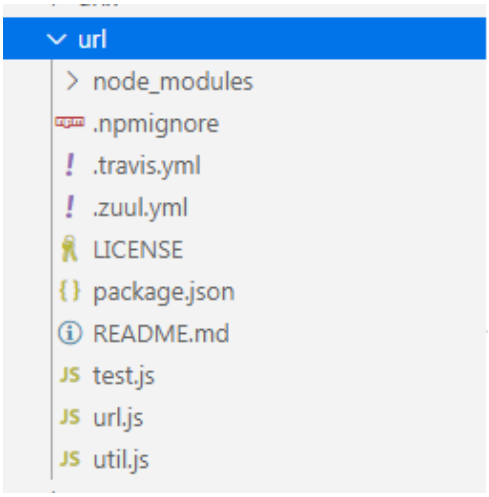


Рис. 4.16 – Модуль URL

Модуль Chai має кілька інтерфейсів, які дозволяють розробнику вибрати найбільш комфортний (Рис. 4.17). BDD (Behavior Driven Development) підтримує та надає виразний стиль мови та читабельність, тоді як стиль TDD (Test Driven Development) забезпечує більш класичний вигляд. Chai доступний як для node.js, так і для браузера і сумісний з будь-яким потрібний тестовим фреймворком. Є можливість написати плагін для перевірки вхідних даних, затвердити перевірку схеми на об'єкті або забезпечити належну поведінку на елементі DOM. API є досить гнучким, що будь-які синхронні завдання можуть бути легко інкапсульовані в рамках одного твердження та повторно використані протягом тестів.

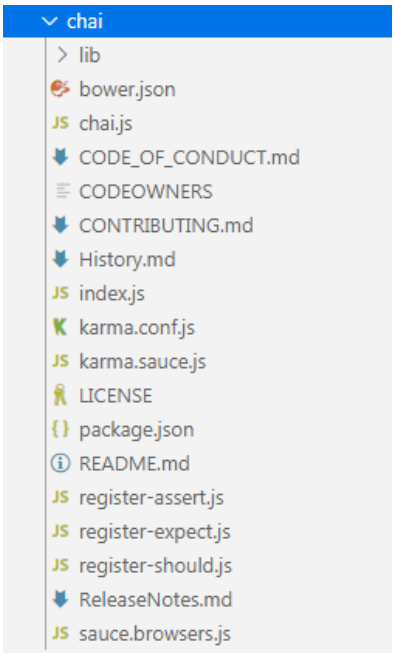


Рис. 4.17 – Модуль Chai

Модуль Nightmare - це високорівнева браузерна автоматизована бібліотека. Мета полягає в тому, щоб розкрити кілька простих методів, що імітують дії користувача (наприклад, goto, type та click), з API, який бачить синхронність для кожного блоку сценаріїв (Рис. 4.18).

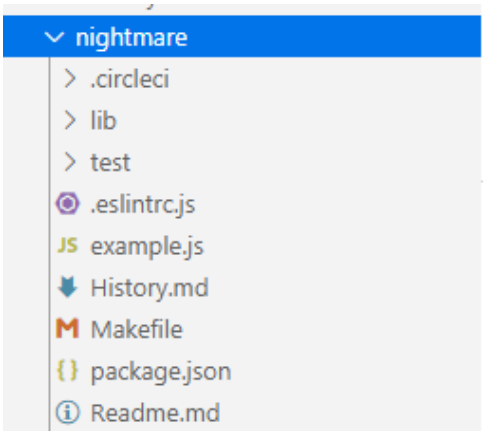


Рис. 4.18 – Модуль Nightmare

Модуль Open активно підтримується та виправляє більшість відкритих вихідних проблем, також він підтримує WSL (Windows Subsystem for Linux) шляхи до програм Windows.

Після проходження всіх тестів кардіологічного веб-додатку користувачеві відображається тест-репорт на основі тестових наборів у вигляді веб-сторінки (Рис. 4.19), а саме діаграма часу проходження кожного набору до загальної кількості часу, графік часу проходження, кількість непройдених тестів по кожному набору, детальний опис непройдених тестів, загальна кількість тестів, діаграма часу проходження тестів у одному наборі та багато іншої потрібної інформації.

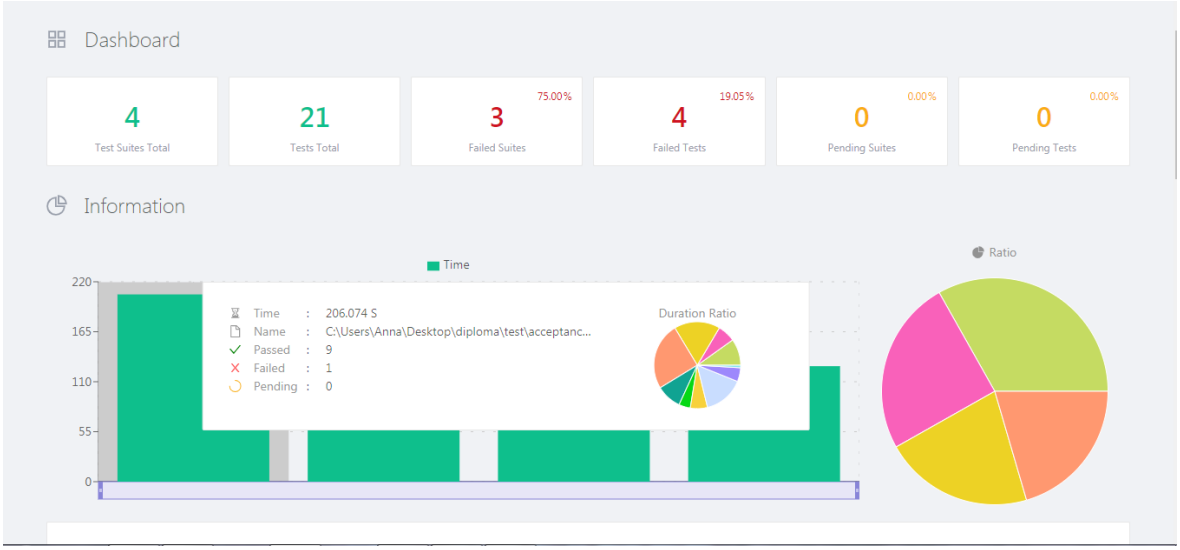


Рис. 4.19 – Сформований тест-репорт до тестування кардіологічного веб-додатку

# 5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Створена система автоматизованих тестів повинна допомогти розробникам кардіологічного веб-додатку швидко та ефективно реагувати на помилки, менше витрачати час на пошук документальної інформації про некоректні результати роботи ресурсу.

## 5.1 Системні вимоги

Для забезпечення корректної роботи автотестів потрібні такі мінімальні системні характеристики:

1. ОС: Windows 7.
2. Процесор: AMD Athlon 64 | Intel Pentium D.
3. Оперативна пам'ять: 2GB.
4. Відеокарта: AMD Radeon HD 2400 | GeForce 9600.
5. Місце на диску: 2Gb.
6. Мережа: високошвидкісне підключення підключення до інтернету.

## 5.2 Робота користувача з програмним продуктом

Для роботи зі створеною системою автоматизованого тестування користувач повинен перейти до директорії в якій розташована ця система, відкрити командну консоль та ввести команду “npm test”.

Після виконання даної команди почнеться процес автоматизованої перевірки запрограмованих тест-кейсів. В залежності з початком виконання певного тест-кейсу буде автоматично відкриватися відповідне вікно, де будуть виконуватися запрограмовані раніше функції.

Всі тест-кейси з Табл. 4.12, 4.13, 4.14 запрограмовані для перевірки кардіологічного веб-додатку. Розглянемо на прикладі перший, продубльований у Табл.5.1 для частини функціоналу входу.

Табл. 5.1 – Тест-кейс для частини функціоналу входу

| Номер | Кейс | Результат |
|---|---|---|
| 1 | Користувач натискає на кнопку входу у власний аккаунт під назвою "Log in" | У користувача з'являється сторінка входу у власний аккаунт з адресою "https://app.cardiolyse.com/sign-in" з центральним блоком "Sign in", який містить поля "Email" та "Password", неактивну кнопку "Sign in", у правому верхньому кутку кнопка "Sign up" для реєстрації та у лівій частині сторінки кнопка "Live demo", яка дає можливість користування програмним продуктом, поки користувач ще не має аккаунта. |

Згідно тест-кейсу під номером 1 з Табл. 5.1 користувач повинен натиснути на кнопку входу у власний аккаунт під назвою "Log in", як це показано на Рис. 5.2, червоним кольором виділено потрібну нам кнопку.

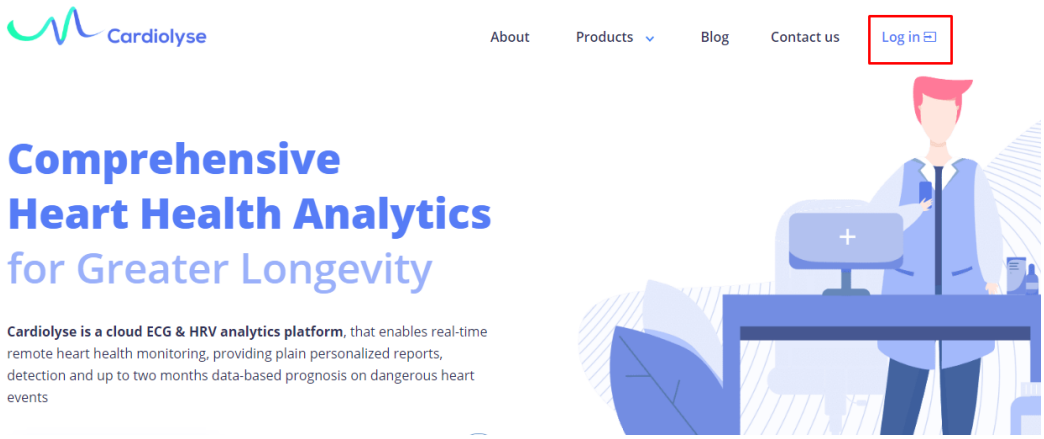


Рис. 5.2 – Автоматичне виконання кейсу під номером 1

Після автомтичного натискання кнопки “Log in” програма переходить до сторінки з адресою "https://app.cardiolyse.com/sign-in" з центральним блоком "Sign in", який містить поля "Email" та "Password", неактивну кнопку "Sign in" (Рис. 5.3).

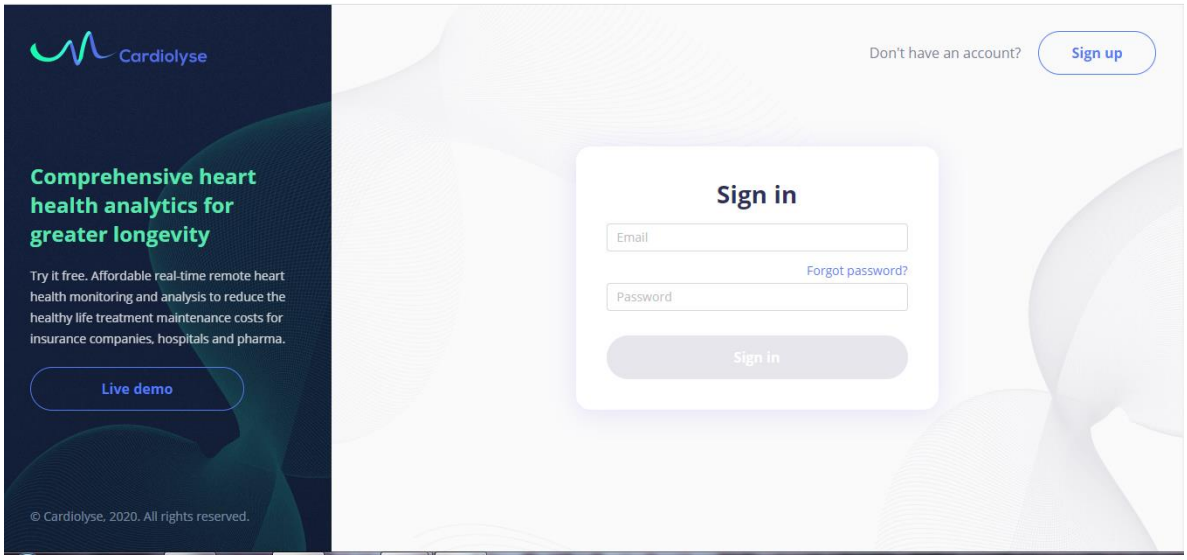


Рис. 5.3 – Отриманий перехід у тест-кейсі під номером 1

Після завершення виконання кейсу перевіряється чи співпадають вказана адреса з очікуваним результатом. В консолі по мірі виконання тест-кейсів виводяться результати їх проходження (Рис. 5.4), а саме їх коротка назва та відповідність результату до очікуваних подій.

```
Log in
  √ should click on login button (8476ms)

Sign in
  √ click email field (6027ms)
  √ click email field and enter valid email (6816ms)
  √ click email field and enter invalid email (6498ms)
  √ click email field and enter invalid email (6299ms)


5 passing (34s)
```

Рис. 5.4 – Результати виконаних всіх тест-кейсів з  Табл. 5.1

Також у консолі виводится кількість тест-наборів, в яких тести успішно пройшли перевірку та у яких було виявлено помилку, кількість самих тестів та час проходження усіх тестів (Рис. 5.5).

```
Test Suites: 2 failed, 2 passed, 4 total
Tests:       3 failed, 18 passed, 21 total
Snapshots:   0 total
Time:        378.921 s
Ran all test suites matching /test\\acceptance/i.
```

Рис. 5.5 – Кількість виконаних та непройдених тестів

Результати також формуються у вигляді веб-сторінки, яку можна відкрити за посиланням https://cardiolyse.herokuapp.com/html-report/report.html у хмарному середовищі Heroku (Рис. 5.6).

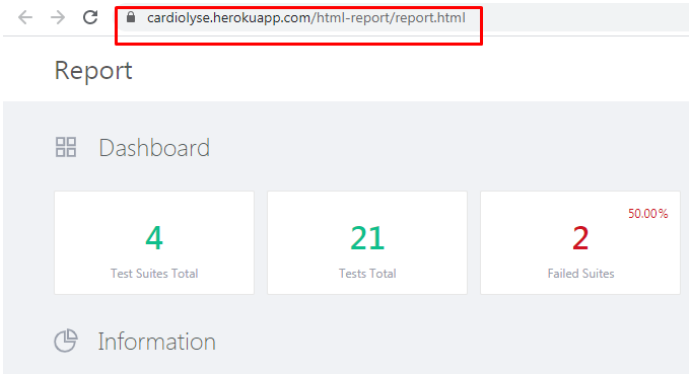


Рис. 5.6 – Посилання, за яким відкривається тест-репорт

Відкривши тест-репорт, користувач має змогу побачити верхню кількісно-описову панель (Рис. 5.7), її виділено червоним. На панелі виводиться:

1. Кількість тест-наборів.
2. Загальна кількість тестів.
3. Кількість наборів, у яких було виявлено помилки.
4. Загальна кількість помилок.
5. Кількість наборів, у яких тести не дочекались відповіді від серверу кардіологічного веб-додатку за встановлений час.
6. Загальна кількість тестів, які не дочекались відповіді від серверу кардіологічного веб-додатку за встановлений час.

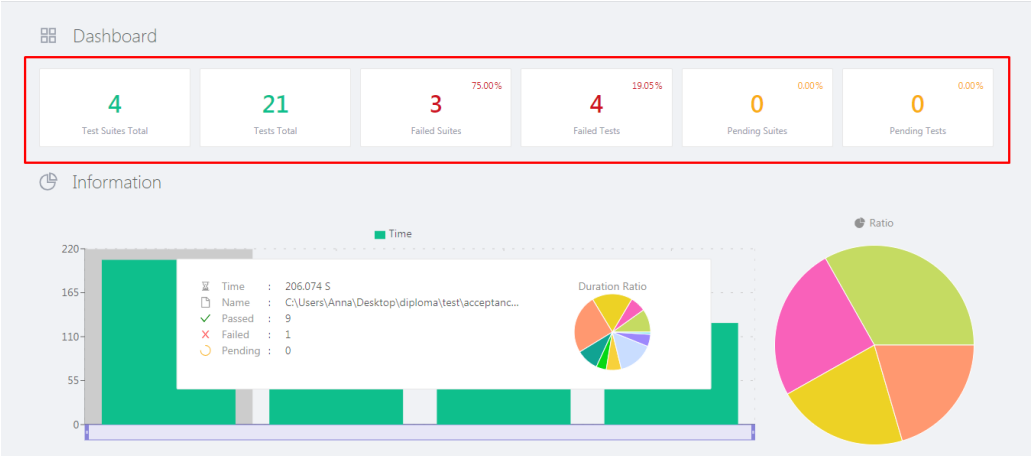


Рис. 5.7 - Кількісно-описова панель

Також в сформованому тест-репорті користувач може відслідкувати час проходження кожного тест-набору. На Рис. 5.8 зображено сформований часовий графік, у якому шкала поділяється на секунди. На графіку зображені стовбці, по висоті яких ми можемо визначити час проходження кожного тест-набору, стовбці розташовуються зліва – направо у відповідності до порядкового номеру їх проходження, тобто якщо тест-набір “Реєстрація” був першим у черзі тестування, то на діаграмі він буде першим відповідно.

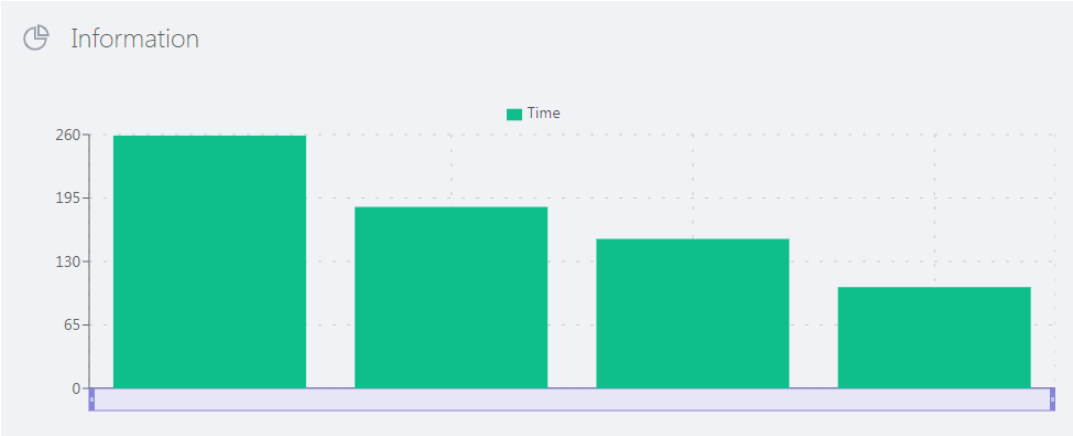


Рис. 5.8 – Часовий графік тестування

Якщо користувач хоче дізнатися скількі часу займав кожен тест у співвідношенні до кожного тест-набору, то він має змогу навести на стовбець бажаного тест-набору, в результаті висвічуєтья поп-ап з точним часом проходження набору, ім’я набору, кількість пройдених у ньому тестів, кількість помилок, кількість тестів, які не дочекались відповіді від серверу кардіологічного веб-додатку за встановлений час та діаграму співвідношення часу. На Рис. 5.9 видно, що загальна кількість тестів для набору “Реєстрація” 10, тобто поділок у діаграмі також 10.

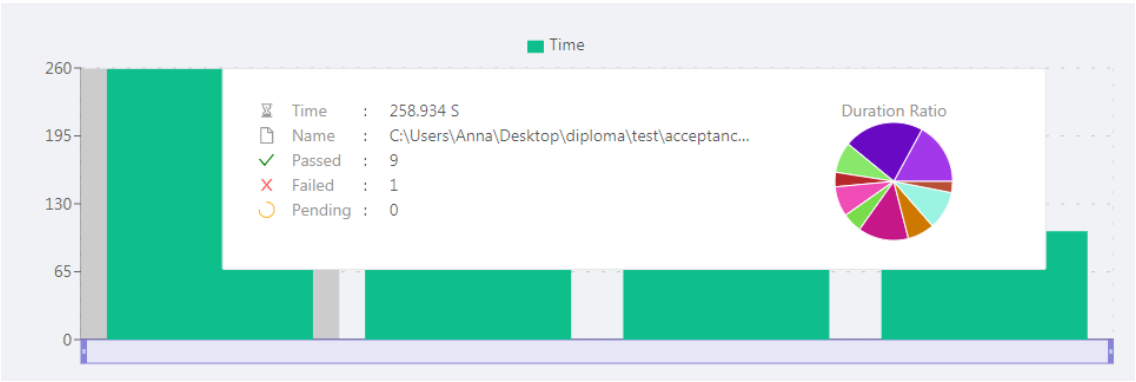


Рис. 5.9 – Часова діаграма тестування кожного кейсу з тест-набору “Реєстрація”

При кліці на стовбець чи частину діаграми сторінка автоматично проскролиться до потрібного тест-набору чи тест-кейсу. Біля графіку часу

проходження кожного тест-набору розміщується діаграма співвідношення часу проходження окремого тест-набору до усього часу (Рис. 5.10). При кліці на частини цієї діаграми сторінка також автоматично проскролиться до обраного нами тест-набору.

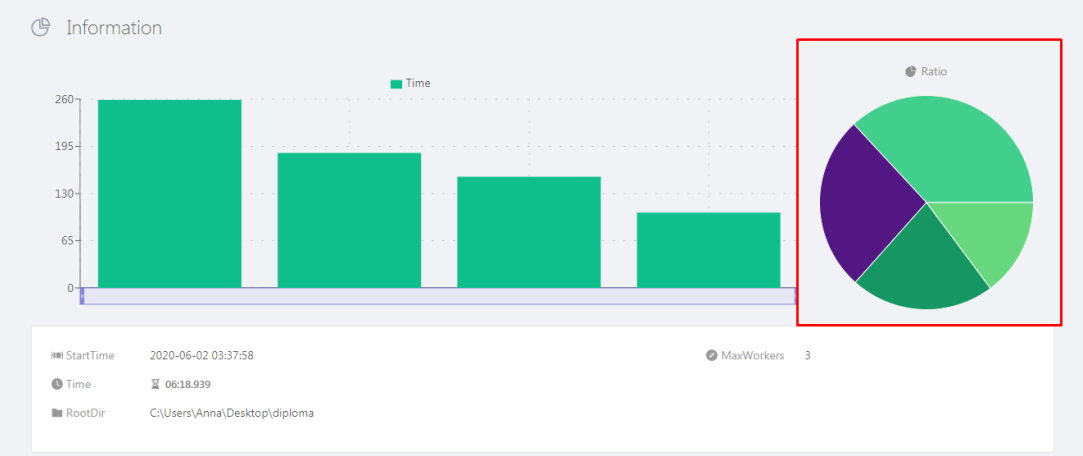


Рис. 5.10 – Часова діаграма усіх тест-наборів

Під блоком діаграми та графіку користувач може продивитися більш детальну інформацію по проходженню тест-наборів (Рис. 5.11), а саме: назву набору, час його проходження, статус проходження тестів у ньому, наслідок. Червоним позначені набори, в яких тести виявили помилки у функціоналі кардіологічного веб-додатку.

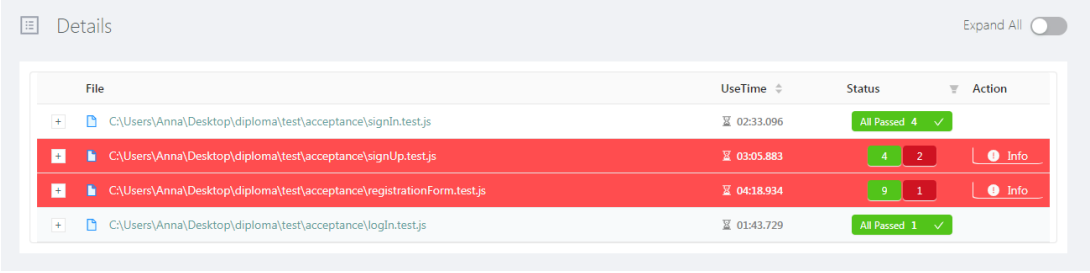


Рис. 5.11 – Блок “Деталі”

Також у блоці “Деталі” можна фільтрувати записи по витраченому часу та статусу тестів у кожному тест-наборі (Рис. 5.12) .

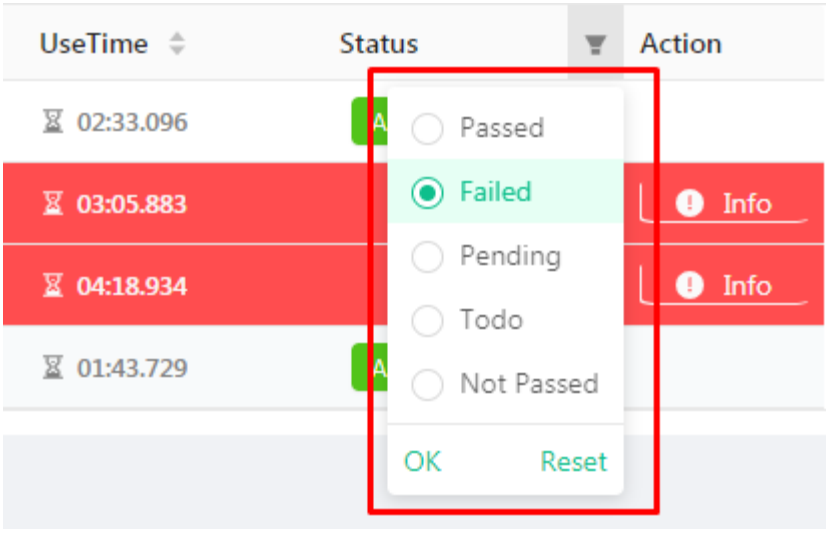


Рис. 5.12 – Фільтр по статусу тестів

Перетянувши ричажок “Expand All” у користувача під кожним блоком тест-наборів автоматично з’являться всі тест-кейси цих наборів, червоним позначені тести, які виявили помилки у функціоналі кардіологічного веб-додатку (Рис. 5.13).

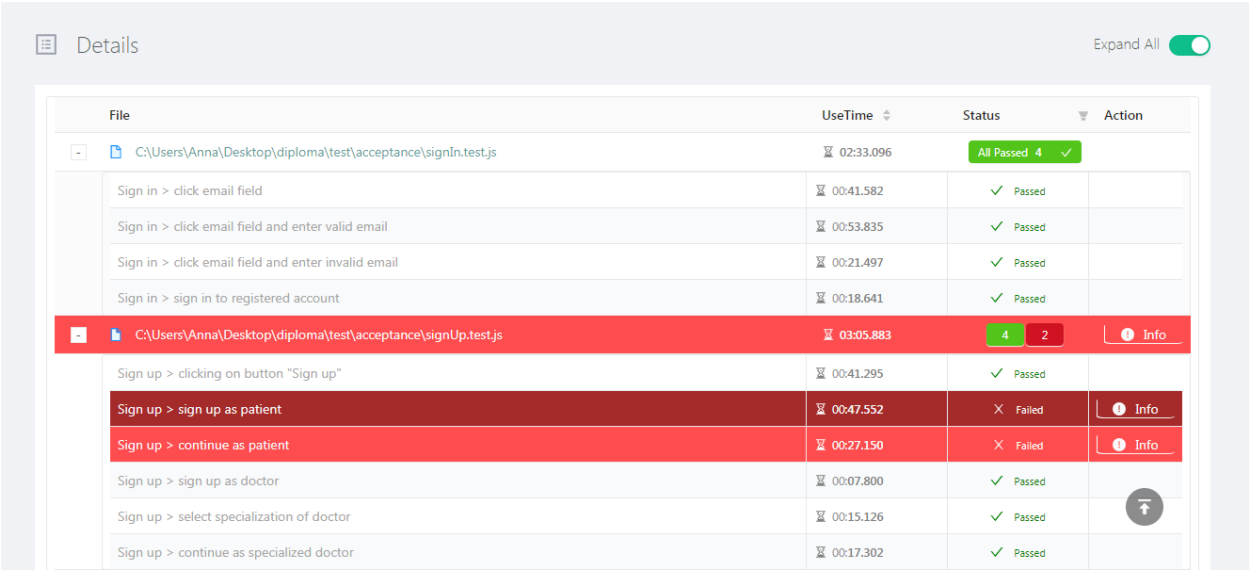


Рис. 5.13 – Перелік усіх тест-кейсів під тест-наборами

Біля кожного тест-кейсу в колонці “UseTime” позначається витрачений час саме на цей тест, статус тесту у колонці “Status” і кнопка, при натисканні

на яку з’являється детальна інформація у разі виявлення помилки, у колонці “Action” (Рис. 5.14)

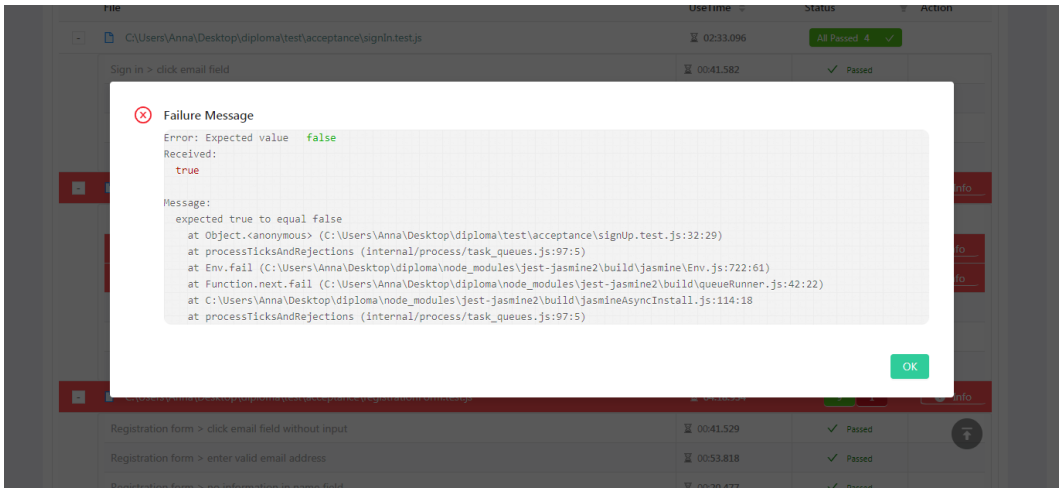


Рис. 5.14 – Детальна інформація у тестах з виявленними помилками

# ВИСНОВКИ

Виконуючи дану роботу, було створено систему автоматизованого тестування кардіологічного веб-додатку. Систему було створено за допомогою інструментів мови JavaScript, бібліотеки Nightmare, Mocha, Chai. Також використовувалися засоби Chrome DevTools, технолії HTML з додаванням CSS.

Систему автоматизованих тестів можна використовувати для тривалої повторюваної перевірки частин програми, коли треба уникнути “людського фактору”. Ця система допоможе проводити набагато швидше тестування, в порівнянні з мануальним. За допомогою автоматизованого тестування веб-додаток матиме змогу проводити навантажувальне тестування, що допоможе вирішити питання оптимізації додатку.

Для роботи зі створеною системою необхідно мати комп’ютер середньої потужності.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. СЕРЦЕВО-СУДИННІ ЗАХВОРЮВАННЯ. ЗАГАЛЬНА ІНФОРМАЦІЯ [Електронний ресурс] – Режим доступу до ресурсу: http://health.medlib.dp.gov.ua/?page_id=976.
2. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Visual_Studio_Code.
3. КОНСОЛЬ В БРАУЗЕРЕ CHROME [Електронний ресурс] – Режим доступу до ресурсу: https://qaevolution.ru/konsol-v-brauzere-chrome/.
4. QAInfo [Електронний ресурс] – Режим доступу до ресурсу: https://www.quality-assurance-group.com/oglyad-instrumentiv-rozrobnyka-u-najpopulyarnishyh-brauzerah/.
5. Серце без збоїв: як запобігти серцево-судинні захворювання? [Електронний ресурс] – Режим доступу до ресурсу: http://www.ffoms.ru/news/monitoring-smi/serdtse-bez-sboev-kak-predotvratit-serdechno-sosudistye-zabolevaniya/.
6. Кардіологічні патології і COVID-19: як зберегти життя пацієнта? [Електронний ресурс] – Режим доступу до ресурсу: http://amnu.gov.ua/kardiologichni-patologiyi-i-covid-19-yak-zberegty-zhyttya-pacziyenta/.
7. The Standard 12 Lead ECG [Електронний ресурс] – Режим доступу до ресурсу: https://ecg.utah.edu/lesson/1.
8. Deaths from cardiac arrests have surged in New York City [Електронний ресурс] – Режим доступу до ресурсу: https://www.economist.com/graphic-detail/2020/04/13/deaths-from-cardiac-arrests-have-surged-in-new-york-city.
9. COVID-19 і кардіологія [Електронний ресурс] – Режим доступу до ресурсу: http://www.webcardio.org/covid-19-i-kardiologhiya.aspx.
10. Heroku [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Heroku.

# Додаток 1

Автоматизоване тестування кардіологічних web – додатків в хмарному середовищі

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ62_62210 20Б 81-1

Аркушів 1

Київ 2020

| Позначення | Найменування | Відмітки |
|---|---|---|
| Документація | | |
| УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТМ62_62210 20Б 81-1 | Записка.docx | Пояснювальна записка |
| Компоненти | | |
| УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТМ62_62210 20Б 12-1 | Опис програмного модулю | Основні компоненти |
| УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТМ62_62210 20Б 12-2 | registrationForm.test.js | Основні компоненти |
| УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТМ62_62210 20Б 13-1 | Теза | Основні компоненти |

# Додаток 2

Автоматизоване тестування кардіологічних web – додатків в хмарному середовищі

Опис програмного модулю

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ62_62210 20Б 12-1

Аркушів 8

Київ 2020

# АНОТАЦІЯ

Додаток містить опис системи автоматизованого тестування кардіологічних web – додатків в хмарному середовищі.

В додатку виконуються такі функції:

— тестування частин кардіологічного web-додатку “Авторизація”, “Реєстраційна форма”, “Реєстрація ролі”, “Інформаційна сторінка пацієнта”;

— формування тест-репорту на основі даних при перевірці;

— Генерація звіту у хмарне середовище Heroku.

# ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ ..... 76
ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ ..... 77
ОПИС ЛОГІЧНОЇ СТРУКТУРИ ..... 78
ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ ..... 79
ВИКЛИК І ЗАВАНТАЖЕННЯ ..... 80

# ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку описано систему автоматизованого тестування кардіологічного веб-додатку.

Модулі системи описані в ДОДАТКУ З.

Систему було створено за допомогою інструментів мови JavaScript, бібліотеки Nightmare, Mocha, Chai. Також використовувалися засоби Chrome DevTools, технології HTML з додаванням CSS. Для відображення звіту у хмарному середовищі була використана хмарна платформа Heroku.

# ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Систему автоматизованих тестів можна використовувати для тривалої повторюваної перевірки частин програми, коли треба уникнути “людського фактору”. Ця система допоможе проводити набагато швидше тестування, в порівнянні з мануальним. За допомогою автоматизованого тестування веб-додаток матиме змогу проводити навантажувальне тестування, що допоможе вирішити питання оптимізації додатку.

# ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Реалізована система автоматизованого тестування перевіряє кардіологічний web-додаток на можливі помилки. Завдяки можливості формувати тест-репорт після перевірки функціоналу додатку користувач матиме наочний інтерфейс, що допоможе швидко дізнатись, чому і де виникли помилки та побачити, де можна оптимізувати систему.

Для функціонування системи потрібно встановити всі необхідні модулі та запустити командою npm test.

# ВИКОРИСТОВУВАНІ ТЕХІЧНІ ЗАСОБИ

Для написання системи автоматизованого тестування я обрала мову JavaScript та серверне оточення побудоване на ній Node.js, фреймворк для тестування Nightmare, мову розмітки HTML та CSS, їх селектори дізнавалась за допомогою інструментів розробника DevTools, середовище розробки Visual Studio Code, хмарну платформу Heroku.

Система функціонує на операційних системах, таких як Windows7, Windows8 та Windows10.

# ВИКЛИК І ЗАВАНТАЖЕННЯ

Для функціонування програми необхідно встановити модулі, які описані у файлі package.json. Далі запустити програму з командного рядку командою npm test. Виконавши ці дії, користувач генерує запуск автоматизованого тестування та матиме можливість перейти за посиланням https://cardiolyse.herokuapp.com/html-report/report.html. На веб-сторінці буде відображено тест-репорт, що допоможе користувачу дізнатись, як функціонує кардіологічний веб-додаток.

# Додаток 3

Автоматизоване тестування кардіологічних web – додатків в хмарному середовищі

Текст програмного модулю

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ62_62210 20Б 12-2

Аркушів 12

Київ 2020

```
const http = require('http')
const fs = require('fs')

async function createReportServer(pathToFile, port) {
  await fs.readFile(pathToFile, function (err, html) {
    if (err) {
      throw err;
    }
    http.createServer(function (request, response) {
      response.writeHeader(200,  {  "Content-Type":  "text/html"
});
      response.write(html);
      response.end();
    }).listen(port);
  });
}

export { createReportServer }

const open = require('open');
const { createReportServer } = require('./createReportServer')
const { BASE_URL } = require('./visitPage')

async function openReport() {
  const pathToFile = './html-report/report.html'

  await createReportServer(pathToFile, process.env.PORT || 8000)
  await open(BASE_URL, { app: ['chrome'] })
}

export { openReport }

const Nightmare = require('nightmare')
const url = require('url')

const BASE_URL = url.format({
  protocol: process.env.PROTOCOL || 'http',
  hostname: process.env.HOST || 'localhost',
  port: process.env.PORT || 8000
})

const DEBUG = process.env.DEBUG || false

function visitPage(path) {
  const location = url.resolve(BASE_URL, path)

  const nightmare = Nightmare({
    show: false, height: 768, width: 1366
  })

  return nightmare.goto(location)
}
```

```
export { visitPage, BASE_URL }

const chai = require('chai')
const expect = chai.expect
const { visitPage } = require('../helpers/visitPage')

describe('Log in', () => {
  test('should click on login button', async function () {
    const page = visitPage('https://cardiolyse.com')

    const url = await page.wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .url()

    expect(url).to.equal('https://app.cardiolyse.com/sign-in')
  })
})

const { openReport } = require('../helpers/openReport')
const chai = require('chai')
const expect = chai.expect
const { visitPage } = require('../helpers/visitPage')

describe('Registration form', () => {
  test('click email field without input', async function () {
    let page = visitPage('https://cardiolyse.com')

    const invalidEmail = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div:nth-child(1) > div > div > span > input', '')
      .exists('#__next > section > main > div > div > form >
div:nth-child(1) > div > div > div')

    expect(invalidEmail).to.equal(true)
  })
```

```
  test('enter valid email address', async function () {
    let page = visitPage('https://cardiolyse.com')

    const validEmail = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div:nth-child(1) > div > div > span > input', 'hi@music.com')
      .exists('#__next > section > main > div > div > form >
div:nth-child(1) > div > div > div')

    expect(validEmail).to.equal(false)
  })

  test('no information in name field', async function () {
    let page = visitPage('https://cardiolyse.com')

    const invalidName = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > input',
'')
```

```
      .exists('#__next > section > main > div > div > form >
div.fields-row > div.ant-row.ant-form-item.ant-form-item-with-
help > div > div > div')

    expect(invalidName).to.equal(true)
  })

  test('name is too short', async function () {
    let page = visitPage('https://cardiolyse.com')

    const invalidName = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > input',
'a')
      .exists('#__next > section > main > div > div > form >
div.fields-row > div.ant-row.ant-form-item.ant-form-item-with-
help > div > div > span > input')

    expect(invalidName).to.equal(true)
  })

  test('numbers in name field', async function () {
    let page = visitPage('https://cardiolyse.com')

    const invalidName = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
```

```
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > input',
'1')
      .exists('#__next > section > main > div > div > form >
div.fields-row > div.ant-row.ant-form-item.ant-form-item-with-
help > div > div > div')

    expect(invalidName).to.equal(true)
  })

  test('cyrillic letters in name field', async function () {
    let page = visitPage('https://cardiolyse.com')

    const invalidName = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > input',
'1')
      .exists('#__next > section > main > div > div > form >
div.fields-row > div.ant-row.ant-form-item.ant-form-item-with-
help > div > div > div')

    expect(invalidName).to.equal(true)
  })

  test('numbers with letters in name field', async function () {
    let page = visitPage('https://cardiolyse.com')

    const invalidName = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
```

```
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > input',
'123ann')
      .exists('#__next > section > main > div > div > form >
div.fields-row > div.ant-row.ant-form-item.ant-form-item-with-
help > div > div > div')

    expect(invalidName).to.equal(true)
  })

  test('valid input in name field', async function () {
    let page = visitPage('https://cardiolyse.com')

    const validName = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > input',
'ann')
      .wait('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > span')
      .visible('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > span')
```

```
    expect(validName).to.equal(true)
  })

  test('enter more 20 symbols in name field', async function ()
{
    let page = visitPage('https://cardiolyse.com')

    const invalidName = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(1) > div > div > span > input',
'annnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn')
      .exists('#__next > section > main > div > div > form >
div.fields-row > div.ant-row.ant-form-item.ant-form-item-with-
help > div > div > div')

    expect(invalidName).to.equal(true)
  })

  test('no information in surname field', async function () {
    let page = visitPage('https://cardiolyse.com')

    const invalidSurname = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
```

```
      .click('#__next > section > main > div > div > div >
button')
      .type('#__next > section > main > div > div > form >
div.fields-row > div:nth-child(2) > div > div > span > input',
'')
      .exist('#__next > section > main > div > div > form >
div.fields-row > div.ant-row.ant-form-item.ant-form-item-with-
help > div > div > div')

    expect(invalidSurname).to.equal(true)
  })
})

const { openReport } = require('../helpers/openReport')
const chai = require('chai')
const expect = chai.expect
const { visitPage } = require('../helpers/visitPage')

describe('Sign in', () => {
  test('click email field', async function () {
    let page = visitPage('https://cardiolyse.com')

    let emailField = await page
      .click('#menu-item-6855 a')
      .type('#__next > section > main > div > div > form >
div.ant-row.ant-form-item > div > div > span > input', '')
      .exists('#__next > section > main > div > div > form >
div:nth-child(1) > div > div > div')

    expect(emailField).to.equal(true)
  })

  test('click email field and enter valid email', async function
() {
    let page = visitPage('https://cardiolyse.com')

    let emailField = await page
      .click('#menu-item-6855 a')
      .type('#__next > section > main > div > div > form >
div.ant-row.ant-form-item > div > div > span > input',
'example@gmail.com')
      .exists('#__next > section > main > div > div > form >
div.ant-row.ant-form-item > div > div > span > span')

    expect(emailField).to.equal(true)
  })

  test('click email field and enter invalid email', async
function () {
    let page = visitPage('https://cardiolyse.com')

    let emailField = await page
```

```
      .click('#menu-item-6855 a')
      .type('#__next > section > main > div > div > form >
div.ant-row.ant-form-item > div > div > span > input', 'do not
correct')
      .wait('#__next > section > main > div > div > form >
div.ant-row.ant-form-item.ant-form-item-with-help > div > div >
div')
      .exists('#__next > section > main > div > div > form >
div.ant-row.ant-form-item.ant-form-item-with-help > div > div >
div')

    expect(emailField).to.equal(true)
  })

  test('sign in to registered account', async function () {
    let page = visitPage('https://cardiolyse.com')

    let url = await page
      .click('#menu-item-6855 a')
      .type('#__next > section > main > div > div > form >
div.ant-row.ant-form-item > div > div > span > input',
'hi@music.com')
      .type('#__next > section > main > div > div > form >
div.forgot_container > div > div > div > span > input',
'annaanna')
      .click('#__next > section > main > div > div > form >
button.ant-btn.Button-ol2u9i-0.KBXpd.ant-btn-primary.ant-btn-
block')
      .wait('#__next > section > section > main > section >
div.header > h2')
      .url()

    expect(url).to.equal('https://app.cardiolyse.com/patients')
  })
})

const { openReport } = require('../helpers/openReport')
const chai = require('chai')
const expect = chai.expect
const { visitPage } = require('../helpers/visitPage')

describe('Sign up', () => {
  test('clicking on button "Sign up"', async function () {
    let page = visitPage('https://cardiolyse.com')

    const url = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
```

```
      .wait('#__next > section > main > div > div')
      .url()

    expect(url).to.equal('https://app.cardiolyse.com/sign-up')
  })

  test('sign up as patient', async function () {
    let page = visitPage('https://cardiolyse.com')

    const disabledItem = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
div.item.item__disabled')
      .visible('#__next > section > main > div > div > div >
div.item.item__disabled')

    expect(disabledItem).to.equal(false)
  })

  test('continue as patient', async function () {
    let page = visitPage('https://cardiolyse.com')

    const disabledItem = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
div.item.item__disabled')
      .click('#__next > section > main > div > div > div >
div.item.item__disabled')
      .visible('#__next > section > main > div > div > div >
div.item.item__disabled')

    expect(disabledItem).to.equal(false)
  })

  test('sign up as doctor', async function () {
    let page = visitPage('https://cardiolyse.com')

    const disabledItem = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
```

```
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1) > span')
      .visible('#__next > section > main > div > div > div >
div:nth-child(1) > span')

    expect(disabledItem).to.equal(true)
  })

  test('select specialization of doctor', async function () {
    let page = visitPage('https://cardiolyse.com')

    const enabledItem = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .visible('#__next > section > main > div > div > div >
button')

    expect(enabledItem).to.equal(true)
  })

  test('continue as specialized doctor', async function () {
    let page = visitPage('https://cardiolyse.com')

    const enabledItem = await page
      .wait('#menu-item-6855 a')
      .click('#menu-item-6855 a')
      .wait('#__next > section > main > div > header > a >
button')
      .click('#__next > section > main > div > header > a >
button')
      .wait('#__next > section > main > div > div > div >
button')
      .click('#__next > section > main > div > div > div >
button')
```

```
      .wait('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
div:nth-child(1)')
      .click('#__next > section > main > div > div > div >
button')
      .visible('#__next > section > main > div > div')

    expect(enabledItem).to.equal(true)
  })
})
```

# Додаток 4

Автоматизоване тестування кардіологічних web – додатків в хмарному середовищі

Теза XX МІЖНАРОДНОЇ СТУДЕНТСЬКОЇ НАУКОВО – ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТМ62_62210 20Б 13-1

Аркушів 3

Київ 2020

В наш час захворювання серцево-судинної системи займають одну з головних позицію серед патологій, що призводять до смертельного наслідку, тож діагностика та точність лікування головна задача сьогодення.

Для більш детального вивчення цієї галузі почали використовувати інформаційні технології. В сучасному світі вплив навколишнього середовища на здоров’я людини, а саме на серцево-судинну систему, дуже сильний. Отже, постає думка про нестаток ресурсу лікарів. Для вирішення цієї проблеми кожна людина може самостійно стежити за станом своєї серцево-судинної системи за допомогою спеціалізованих додатків. Моніторячи самостійно аналізи можна зекономити лікарям час діагностики пацієнта і дати час на обмірковування найбільш ефективного способу лікування. Але, нажаль, такі технології не мають довіри користувачів, тож не використовуються широко.

На основі зібраних аналізів лікарі можуть заключати діагнози, тож збережені дані не повинні мати похибки. Для усунення всіх погрішностей в таких програмах особливо важливо все ретельно перевіряти. Саме для таких потреб почали використовувати тестування в розробці.

Головна мета тестування в кардіологічному додатку це перевірити правильність та вчасність запису цих даних, адже невірно записані дані скасовують всі переваги використання новітніх технологій. В сьогоденні існує великий вибір засобів для тестування, але зупинимося та розглянемо детальніше Nightmare.

Чому Nightmare?

По-перше, Nightmare підтримує JavaScript, Python, Java, C #, Ruby та Perl – основні мови програмування для автоматизації тестування програмного забезпечення. Цей інструмент зможе за короткий час перетворити скрипти, написані на будь-якій з цих мов програмування в сумісний код Nightmare.

По-друге, головною перевагою Nightmare є його доступність відкритиого коду. Завдяки цій здатності більшість інженерів із автоматизації якості використовують Nightmare для тестування продукції своєї компанії.

По-третє, в наш час сервери розташовані на різних операційних системах, кардіологічні додатки не виняток, тому в цьому випадку нам потрібна ще одна перевага Nightmare - кросплатформний. Він може підтримуватися та працювати в різних операційних системах, таких як Windows, Linux, Mac OS, UNIX тощо.

Наразі багато людських життів залежать від точності запрограмованих алгоритмів, які прискіпливо перевіряються та не мають дати збою. Тестування це перевірка правильності роботи алгоритмів та взаємодії всіх компонентів між собою. Перевірка виконується не на реальних прикладах чи живих людях, а програмно, що є цілком безпечно. Добре перевірений продукт має більший коефіцієнт довіри, як результат частіше використовується.

Отже, людські помилки можуть призвести до дефектів у будь-якій галузі розробки програмного забезпечення, і наслідки цього можуть бути самими різними - від незначних до катастрофічних. Хочу зазначити, що тестування може мати великий вплив, коли мова йде про тестування даних пов’язаних з здоров'ям людини.